(19) BUNDESREPUBLIK DEUTSCHLAND

DEUTSCHES PATENT- UND MARKENAMT

(12) Übersetzung der europäischen Patentschrift

(97) EP 0 680 652 B 1

(10) DE 694 20 547 T 2

(51) Int. Cl.[7]: G 10 L 13/00
G 10 L 5/02
G 10 L 7/02
G 10 L 9/14

(21) Deutsches Aktenzeichen: 694 20 547.8
(86) PCT-Aktenzeichen: PCT/US94/00770
(96) Europäisches Aktenzeichen: 94 907 854.7
(87) PCT-Veröffentlichungs-Nr.: WO 94/17517
(86) PCT-Anmeldetag: 18. 1. 1994
(87) Veröffentlichungstag der PCT-Anmeldung: 4. 8. 1994
(97) Erstveröffentlichung durch das EPA: 8. 11. 1995
(97) Veröffentlichungstag der Patenterteilung beim EPA: 8. 9. 1999
(47) Veröffentlichungstag im Patentblatt: 13. 7. 2000

(30) Unionspriorität:
7621 21. 01. 1993 US

(73) Patentinhaber:
Apple Computer, Inc., Cupertino, Calif., US

(74) Vertreter:
Hössle & Kudlek, 80469 München

(84) Benannte Vertragstaaten:
DE, ES, FR, GB

(72) Erfinder:
NARAYAN, Shankar, Palo Alto, CA 94306, US

(54) WELLENFORM-MISCHUNGSVERFAHREN FÜR SYSTEM ZUR TEXT-ZU-SPRACHE UMSETZUNG

DE 694 20 547 T 2

DE 694 20 547 T 2

03.12.99

694 20 547.8-08 F15 004 EP/DE

Appl Computer, Inc.

Wellenform-Mischungsverfahren für System zur Text-zu-Sprache Umsetzung

Die vorliegende Erfindung betrifft Systeme zur glatten Verkettung bzw. Kettung von quasi-periodischen Wellenformen, wie zum Beispiel kodierte Diphon-Aufzeichnungen, die verwendet werden beim Übersetzen bzw. Umsetzen von Text in einem Computersystem zu synthetisierter Sprache bzw. zu synthetischer Sprachausgabe.

In Text-zu-Sprach-Umwandlungssystemen wird ein in einem Computer gespeicherter Text übersetzt in synthetisierte Sprache. Wie es erkannt wird, könnte diese Art von System weitläufig angewendet werden, wenn es sich kostengünstig bereitstellen ließe. Beispielhaft könnte ein Text-zu-Sprach-System verwendet werden zum entfernten Abfragen elektronischer Nachrichten über eine Telefonleitung, indem der Computer, welcher die elektronische Nachricht gespeichert hat, veranlaßt wird, Sprache zu synthetisieren, die die elektronische Nachricht repräsentiert bzw. wiedergibt. Ferner könnten solche System verwendet werden, um Personen Texte vorzulesen, die sehbehindert sind. In dem Textverarbeitungsumfeld könnten Text-zu-Sprach-Systeme verwendet werden als Unterstützung beim Gegenlesen eines umfangreichen Dokumentes.

Systeme gemäß dem Stand der Technik, die relativ kostengünstig sind, verfügen jedoch über eine Sprachqualität, die relativ schlecht ist, so daß diese unbequem zu verwenden

oder schwierig zu verstehen sind. Um eine gute Sprachqualität zu erhalten, erfordern Sprachsynthetisiersysteme spezielle Hardware, die sehr teuer ist, und/oder eine große Menge an Speicherraum in dem Computersystem, welches den Ton bzw. die Laute erzeugt.

In Text-zu-Sprach-Systemen untersucht ein Algorithmus eine Eingangstextstring bzw. -folge und übersetzt die Worte in der Textfolge in eine Folge von Diphonen, welche in synthetisierte Sprache umgesetzt werden müssen. Text-zu-Sprach-Systeme analysieren auch den Text basierend auf dem Worttyp und dem Kontext, um eine Betonungssteuerung zu generieren, die verwendet wird zum Einstellen der Dauer der Töne bzw. Laute sowie der Tonhöhe bzw. dem Pitch (engl. pitch) der in der Sprache involvierten Laute.

Diphone bestehen aus einer Spracheinheit, die gebildet ist aus dem Übergang zwischen einem Laut oder Phonem und einem benachbarten Laut oder Phonem. Diphone beginnen üblicherweise an der Mitte eines Phonems und enden an der Mitte eines benachbarten Phonems. Dies erhält bzw. konserviert den Übergang zwischen den Lauten relativ gut.

Text-zu-Sprach-Systeme, die auf amerikanischem Englisch basieren, verwenden abhängig von der spezifischen Implementierung etwa 50 unterschiedliche Laute, welche Phone genannt werden. Von diesen 50 unterschiedlichen Lauten verwendet die Normalsprache etwa 1800 Diphone von den möglichen 2500 Phonpaaren. Demzufolge muß ein Text-zu-Sprach-System in der Lage sein, 1800 Diphone zu reproduzieren bzw. wiederzugeben. Um die Sprachdaten unmittelbar für jedes Diphon speichern zu können, wäre ein enormes Ausmaß an Speicher erforderlich. Demzufolge wurden Kompressionstech-

niken entwickelt, um das Ausmaß an erforderlichem Speicher zum Speichern der Diphone zu begrenzen.

Text-zu-Sprach-Systeme gemäß dem Stand der Technik sind unter anderem beschrieben in dem US-Patent Nr. 8,452,168 mit dem Titel COMPRESSION OF STORED WAVE FORMS FOR ARTIFICIAL SPEECH, erfunden von Sprague; und dem US-Patent Nr. 4,692,941 mit dem Titel REAL-TIME TEXT-TO-SPEECH CONVERSION SYSTEM, erfunden von Jacks, et al. Weiterer technischer Hintergrund bezüglich Sprachsynthese kann gefunden werden in dem US-Patent Nr. 4,384,169 mit dem Titel METHOD AND APPARATUS FOR SPEECH SYNTHESIZING, erfunden von Mozer, et al.

Zwei verkettete Diphone verfügen über einen End- bzw. Endungsrahmen und einen Anfangsrahmen. Der Endungsrahmen des linken Diphons muß vermischt bzw. abgemischt werden mit dem Anfangsrahmen des rechten Diphons, ohne daß hörbare Diskontinuitäten oder Klicks bzw. Laute erzeugt werden. Da die rechte Grenze bzw. der rechte Rand des ersten Diphons und der linke Rand des zweiten Diphons in den meisten Situationen demselben Phonem entsprechen, wird angenommen, daß diese in einem Verkettungspunkt ähnlich aussehen. Da jedoch die zwei Diphonkodierungen aus unterschiedlichen Kontexten extrahiert sind, werden diese nicht identisch aussehen. Demzufolge versuchten Vermischungs- bzw. Abmischtechniken gemäß dem Stand der Technik, verkettete Wellenformen an dem Ende und dem Anfang von linkem bzw. rechtem Rahmen jeweils zu mischen. Da Ende und Anfang von Rahmen nicht gut aufeinander abgestimmt sein können, resultiert ein Mischrauschen bzw. ein Mischgeräusch. Eine Kontinuität des Tones zwischen benachbarten Diphonen ist demzufolge gestört.

Trotz der bisherigen Aktivitäten in diesem Umfeld finden die Verwendung von Text-zu-Sprach-Systemen keine weitläufige Akzeptanz. Es ist daher wünschenswert, ein lediglich auf Software beruhendes Text-zu-Sprach-System bereitzustellen, welches auf eine breite Vielzahl von Mikrocomputerplattformen portierbar ist und hohe Sprachqualität bereitstellt und in Echtzeit an solchen Plattformen betrieben werden kann.

Gemäß einem ersten Gesichtspunkt stellt die vorliegende Erfindung eine Vorrichtung bereit zur Verkettung eines ersten digitalen Rahmens von N Proben mit jeweiligen Beträgen bzw. Amplituden (Magnituden, engl. magnitudes), welche eine erste quasi-periodische Wellenform darstellen, mit einem zweiten digitalen Rahmen von M Proben mit jeweiligen Beträgen, welche eine zweite quasi-periodische Wellenform darstellen, umfassend:

einen Puffer zum Speichern der Proben des ersten und zweiten digitalen Rahmens;

Mittel, welche mit dem Pufferspeicher gekoppelt sind, zur Bestimmung eines Mischungspunktes für den ersten und den zweiten digitalen Rahmen, ansprechend auf die Beträge der Proben in den ersten und zweiten digitalen Rahmen bzw. in dem ersten und dem zweiten digitalen Rahmen;

Vermischungsmittel, welche mit dem Pufferspeicher und den Mitteln zur Bestimmung gekoppelt sind, um eine digitale Sequenz zu berechnen, welche eine Verkettung der ersten und zweiten quasi-periodischen Wellenformen darstellt, ansprechend auf den ersten Rahmen, den zweiten Rahmen und den Vermischungspunkt.

Die Technik ist ferner anwendbar auf die Verkettung zweier beliebiger quasi-periodischer Wellenformen, die üblicherweise angetroffen werden in der Ton- bzw. Lautsynthese oder in Sprache, Musik, Toneffekten oder dergleichen.

Gemäß einer Ausführungsform der vorliegenden Erfindung wird das System betrieben, indem zuerst ein erweiterter Rahmen, ansprechend bzw. als Antwort auf den ersten digitalen Rahmen, berechnet wird, wonach eine Teilmenge bzw. ein Untersatz des erweiterten Rahmens gesucht bzw. gefunden wird, welcher relativ gut auf den zweiten digitalen Rahmen paßt bzw. abgestimmt ist. Der optimale Vermischungspunkt wird anschließend definiert als eine Probe in der Teilmenge des erweiterten Rahmens. Die Teilmenge des erweiterten Rahmens, welche relativ gut auf den zweiten digitalen Rahmen paßt, wird bestimmt unter Verwendung einer minimalen Differenzfunktion des mittleren Betrages bezüglich der Proben in der Teilmenge. Der Vermischungspunkt umfaßt in diesem Zusammenhang die erste Probe der Teilmenge. Um die verkettete Wellenform zu erzeugen, wird die Teilmenge des erweiterten Rahmens mit dem zweiten Digitalrahmen kombiniert und mit dem Anfangssegment des erweiterten Rahmens verkettet, um die verkette Wellenform zu erzeugen.

Die verkettete Folge wird anschließend in eine Analogform oder eine andere physikalische Darstellung der gemischten Wellenformen umgewandelt.

Gemäß einem weiteren Gesichtspunkt stellt die vorliegende Erfindung eine Vorrichtung zum Synthetisieren von Sprache, ansprechend auf einen Text, bereit, umfassend:

Mittel zur Übersetzung bzw. Umsetzung von Text in eine Sequenz von Ton- bzw. Lautgsegementcodierungen;

Mittel, welche ansprechend sind auf die Lautsegmentcodierungen in der Sequenz, um die Sequenz der Lautsegmentcodierungen zu decodieren zur Herstellung von Strings von digitalen Rahmen einer Anzahl von Proben, welche Töne bzw. Laute für jeweilige Ton- bzw. Lautsegmentcodierungen in der Sequenz darstellen, wobei die identifizierten Strings der digitalen Rahmen Anfänge und Endungen aufweisen;

Mittel zur Verkettung eines ersten digitalen Rahmens, an dem Ende eines identifizierten Strings von digitalen Rahmen einer bestimmten Lautsegmentcodierung in den Sequenzen, mit einem zweiten digitalen Rahmen an dem Anfang eines identifizierten Strings von digitalen Rahmen einer benachbarten Lautsegmentcodierung in der Sequenz, um eine Sprachdatensequenz zu erzeugen, enthaltend

einen Pufferspeicher zum Speichern der Proben von ersten und zweiten digitalen Rahmen;

Mittel, welche mit dem Pufferspeicher gekoppelt sind, um einen Vermischungspunkt für die ersten und zweiten digitalen Rahmen zu bestimmen, und zwar ansprechend auf Beträge der Proben in den ersten und zweiten digitalen Rahmen; und

Vermischungsmittel, welche mit dem Pufferspeicher und den Mitteln zur Bestimmung gekoppelt sind, um eine digitale Sequenz zu berechnen, welche eine Verkettung der ersten und zweiten Lautsegmente darstellt, und zwar ansprechend auf den ersten Rahmen, den zweiten Rahmen und den Vermischungspunkt; und

einen Audiowandler, der mit den Mitteln zur Verkettung gekoppelt ist, um ansprechend auf die Sprachdatensequenz synthetische Sprache zu generieren.

In einer Ausführungsform der Erfindung enthalten die Betriebsmittel bzw. Ressourcen, welche den optimalen Vermischungspunkt bestimmen, Berechnungsressourcen, welche einen erweiterten Rahmen berechnen, umfassend eine diskontinuitätenglättende Verkettung des ersten Digitalrahmens mit einer Replica bzw. Wiederholung des ersten Digitalrahmens. Weitere Ressourcen finden eine Teilmenge des erweiterten Rahmens mit einer minimalen mittlere Betragsdifferenz zwischen den Proben in der Teilmenge und in dem zweiten Digitalrahmen, und definieren den optimalen Vermischungspunkt als die erste Probe in dieser Teilmenge. Die Mischungsressourcen enthalten Software oder andere Berechnungsressourcen, welche einen ersten Satz an Proben bereitstellen, abgeleitet von dem ersten Digitalrahmen, wobei der Vermischungspunkt als ein erstes Segment der Digitalsequenz vorliegt bzw. angenommen wird. Anschließend wird der zweite digitale Rahmen mit der Teilmenge des erweiterten Rahmens kombiniert, mit Hervorhebung der Teilmenge des erweiterten Rahmens in einer Anfangsprobe und Hervorhebung des zweiten digitalen Rahmens in einer Endprobe, um ein zweites Segment der Digitalsequenz zu erzeugen. Das erste Segment und das zweite Segment werden kombiniert, um eine Sprachdatensequenz darzustellen.

Gemäß noch weiteren bevorzugten Ausführungsformen der Erfindung umfaßt die Text-zu-Sprach-Vorrichtung ein Bearbeitungsmodul, welches ansprechend auf den eingegebenen bzw. Eingangstext die Tonhöhe bzw. den Pitch und die Dauer der identifizierten Strings von Digitalrahmen in der Sprachda-

tensequenz einstellt. Ferner basiert der Dekoder auf einer Vektorquantisierungstechnik, welche ausgezeichnete Kompressionsqualität bereitstellt, wobei sehr geringe Dekodierressourcen erforderlich sind.

Weitere Gesichtspunkte und Vorteile der vorliegenden Erfindung werden ersichtlich beim Lesen der Figuren, der detaillierten Beschreibung von bevorzugten Ausführungsformen, welche lediglich beispielhaft erfolgt, sowie der beiliegenden Ansprüche.

Fig. 1 ist ein Blockdiagramm einer generischen Hardwareplattform, die ein Text-zu-Sprach-System gemäß der vorliegenden Erfindung enthält.

Fig. 2 ist ein Flußdiagramm, welches eine Basis-Text-zu-Sprach-Routine bzw. -Programm der vorliegenden Erfindung darstellt.

Fig. 3 stellt das Format von Diphonaufzeichnungen gemäß einer Ausführungsform der vorliegenden Erfindung dar.

Fig. 4 ist ein Flußdiagramm, welches eine Codiereinrichtung für Sprachdaten gemäß der vorliegenden Erfindung darstellt.

Fig. 5 ist ein Graph, welcher diskutiert wird unter Bezugnahme auf die Abschätzung von Pitchfilterparametern in der in Fig. 4 gezeigten Codiereinrichtung.

Fig. 6 ist ein Flußdiagramm, welches die Vollsuche darstellt, die in der Codiereinrichtung von Fig. 4 verwendet wird.

Fig. 7 ist ein Flußdiagramm, welches einen Decoder für Sprachdaten gemäß der vorliegenden Erfindung darstellt.

Fig. 8 ist ein Flußdiagramm, welches eine Technik zum Mischen bzw. Abmischen des Anfangs und des Endes von benachbarten Diphonaufzeichnungen darstellt.

Fig. 9 besteht aus einem Satz von Graphen, auf die Bezug genommen wird bei der Erläuterung der Vermischungs- bzw. Abmischtechnik von Fig. 8.

Fig. 10 ist ein Graph, welcher ein typisches Diagramm von Pitch gegen die Zeit für eine Sequenz von Rahmen von Sprachdaten darstellt.

Fig. 11 ist ein Flußdiagramm, welches eine Technik darstellt zur Vergrößerung bzw. Anhebung der Pitchperiode für einen bestimmten Rahmen.

Fig. 12 ist ein Satz von Graphen, auf die Bezug genommen wird bei der Erläuterung der in Fig. 11 gezeigten Technik.

Fig. 13 ist ein Flußdiagramm, welches eine Technik zum Absenken bzw. Verringern der Pitchzeit eines bestimmten Rahmens darstellt.

Fig. 14 ist ein Satz von Graphen, auf die bei der Erläuterung der Technik von Fig. 13 Bezug genommen wird.

Fig. 15 ist ein Flußdiagramm, welches eine Technik darstellt zum Einfügen einer Pitchperiode zwischen zwei Rahmen in einer Sequenz.

Fig. 16 ist ein Satz von Kurvenverläufen bzw. Graphen, auf die Bezug genommen wird bei der Erläuterung der Technik von Fig. 15.

Fig. 17 ist ein Flußdiagramm, welches eine Technik darstellt zum Entfernen bzw. Löschen einer Pitchperiode in einer Sequenz von Rahmen.

Fig. 18 ist ein Satz von Graphen, auf die bei der Erläuterung der Technik von Fig. 17 Bezug genommen wird.

Einer detaillierte Beschreibung der bevorzugten Ausführungsformen der Erfindung wird angegeben unter Bezugnahme auf die Figuren. Fig. 1 und 2 stellen eine Übersicht eines Systemes bereit, welches die vorliegende Erfindung beinhaltet. Fig. 3 stellt die grundsätzliche Weise dar, in welcher Diphonaufzeichnungen gemäß der vorliegenden Erfindung gespeichert werden. Fig. 4 bis 6 stellen Codierverfahren dar, die auf einer Vektorquantisierung gemäß der vorliegenden Erfindung basieren. Fig. 7 stellt einen Decodieralgorithmus gemäß der vorliegenden Erfindung dar.

Fig. 8 und 9 stellen eine bevorzugte Technik zum Mischen von Anfang und Ende von benachbarten Diphonaufzeichnungen

dar. Fig. 10 bis 18 stellen die Techniken zur Steuerung des Pitches und der Dauer von Tönen bzw. Lauten in dem Text-zu-Sprach-System dar.

I. Systemüberblick (Fig. 1 bis 3)

Fig. 1 stellt eine Basismikrocomputerplattform dar, die ein Text-zu-Sprach-System beinhaltet, welches erfindungsgemäß auf Vektorquantisierung beruht. Die Plattform umfaßt eine Hauptprozessoreinheit 10, die mit einem Hostsystem 11 gekoppelt ist. Eine Tastatur 12 oder eine andere Texteingabeeinrichtung ist in dem System bereitgestellt. Ferner ist ein Anzeigesystem 13 mit dem Hostsystembus gekoppelt. Das Hostsystem enthält ebenfalls ein nicht-flüchtiges Speichersystem, wie zum Beispiel ein Plattenlaufwerk 14. Ferner enthält das System einen Hostspeicher 15. Der Hostspeicher enthält Text-zu-Sprach-(TTS)-Codes, einschließlich codierter Sprachtabellen, Puffer und anderer Hostspeicher. Der Text-zu-Sprach-Code wird verwendet, um Sprachdaten zu generieren, die einem Audio-Ausgangsmodul 16 zuzuführen sind, welches einen Lautsprecher 17 enthält. Der Code enthält ebenfalls einen optimalen Vermischungspunkt sowie ein Diphonverkettungsprogramm bzw. eine Optimal-Mischpunkt-Diphonverkettungsroutine, wie unter Bezugnahme auf die Fig. 8 und 9 im Detail beschrieben wird.

Gemäß der vorliegenden Erfindung enthalten die codierten Sprachtabellen ein TTS-Wörterbuch, welches verwendet wird zum Übersetzen von Text in einen String von Diphonen. Ferner ist ebenfalls eine Diphontabelle enthalten, die die Diphone in identifizierte Strings von Quantisierungsvektoren übersetzt. Eine Quantisierungsvektortabelle wird verwendet, um die Lautsequenzcodes der Diphontabelle in die

Sprachdaten zur Audioausgabe zu übersetzen. Das System kann auch eine Vektorquantisierungstabelle zur Codierung enthalten, welche bei Bedarf in den Hostspeicher 15 geladen wird.

Die in Fig. 1 dargestellte Plattform repräsentiert ein beliebiges generisches Mikrocomputersystem, einschließlich eines auf Macintosh basierenden Systems, eines auf DOS basierenden Systems, eines auf UNIX basierenden Systems oder anderer Typen von Mikrocomputern. Der Text-zu-Sprach-Code und die codierten Sprachtabellen gemäß der vorliegenden Erfindung zur Decodierung nehmen eine relativ geringe Menge an Hostspeicher 15 ein. Beispielhaft kann ein erfindungsgemäßes Text-zu-Sprach-Decodiersystem implementiert werden, welches weniger als 640 Kilobyte an Hauptspeicher besetzt, wobei dennoch hochqualitative natürlich klingende synthetisierte Sprache erzeugt wird.

Der durch den Text-zu-Sprach-Code ausgeführte Basisalgorithmus ist in Fig. 2 dargestellt. Das System empfängt zuerst den eingegebenen Text bzw. Eingabetext (Block 20). Der Eingabetext wird übersetzt in Diphonstrings unter Verwendung des TTS-Wörterbuches bzw. Dictionary (Block 21). Gleichzeitig wird der Eingabetext analysiert bzw. untersucht, um Betonungssteuerdaten zu erzeugen, um den Pitch und die Dauer der die Sprache bildenden Diphone zu steuern (Block 22).

Nachdem der Text in Diphonstrings übersetzt ist, werden die Diphonstrings dekomprimiert, um vektorquantisierte Datenrahmen zu erzeugen (Block 23). Nachdem die vektorquantisierten (VQ) Datenrahmen gebildet sind, werden die Anfänge und Endungen von benachbarten Diphonen vermischt bzw. abgemischt, um jegliche Diskontinuitäten zu glätten (Block 24).

Anschließend werden die Dauer und der Pitch der Diphon-VQ-Datenrahmen eingestellt, und zwar ansprechend auf die Betonungssteuerdaten (Block 25 und 26). Schließlich werden die Sprachdaten dem Audioausgabesystem zur Echtzeiterzeugung zugeführt (Block 27). Für Systeme mit ausreichender Verarbeitungsleistung kann ein adaptiver Nachfilter angewendet werden, um die Sprachqualität weiter zu verbessern.

Das TTS-Wörterbuch kann implementiert werden unter Verwendung einer beliebigen von einer Vielzahl von in der Technik bekannten Techniken. Gemäß der vorliegenden Erfindung werden Diphonaufzeichnungen in einem stark komprimierten Format implementiert, wie es in Fig. 3 dargestellt ist.

Wie es in Fig. 3 gezeigt ist, sind Aufzeichnungen für einen linken Diphon 30 und eine Aufzeichnung für einen rechten Diphon 31 dargestellt. Die Aufzeichnung für den linken Diphon 30 enthält einen Zählwert 32 von der Anzahl NL der Pitch- bzw. Aufteilungsperioden in dem Diphon. Des weiteren ist ein Zeiger bzw. Pointer 33 enthalten, welcher zu einer Tabelle der Länge NL zeigt, die die Zahl LP_i für jede Aufteilungs- bzw. Pitchperiode speichert, wobei i zwischen 0 und NL-1 der Pitchwerte für entsprechend komprimierte Rahmenaufzeichnungen liegt. Schließlich ist ein Zeiger 34 enthalten, welcher zu einer Tabelle 36 von ML vektorquantisierten komprimierten Sprachaufzeichnungen zeigt, jeweils mit einer festgelegten Satzlänge an kodierter Rahmengröße bezüglich eines Nominalpitches der codierten Sprache für den linken Diphon. Der Nominalpitch basiert auf der gemittelten Anzahl an Proben für eine gegebene Pitchperiode für die Sprachdatenbank.

Eine ähnliche Struktur kann für den rechten Diphon 31 erkannt werden. Unter Verwendung einer Vektorquantisierung ist eine Länge der komprimierten Sprachaufzeichnungen sehr kurz in Bezug auf die erzeugte Sprachqualität.

Das Format der vektorquantisierten Sprachaufzeichnungen kann besser verstanden werden unter Bezugnahme auf die Rahmencodierroutine und die Rahmendecodierroutine, wie folgend unter Bezugnahme auf die Fig. 4 bis 7 beschrieben.

II. Die Encoder/Decoder-Programme bzw. -Routinen (Fig. 4 bis 7)

Die Codierroutine ist in Fig. 4 dargestellt. Der Codierer akzeptiert als Eingabe einen Rahmen s_n von Sprachdaten. Bei dem bevorzugten System sind die Sprachproben dargestellt als 12 oder 16 Bit auf zwei komplementierte Zahlen, gesampelt bzw. abgetastet bei 22,252 Hz. Diese Daten werden aufgeteilt in nichtüberlappende Rahmen s_n mit einer Länge N, wobei N als die Rahmengröße bezeichnet wird. Der Wert von N ist abhängig von dem Nominalpitch der Sprachdaten. Wenn der Nominalpitch der aufgezeichneten Sprache geringer ist als 165 Proben (oder 135 Hz) wird der Wert von N als 96 gewählt. Ansonsten wird eine Rahmengröße von 160 verwendet. Der Codierer transformiert die N-Punkt Datensequenz s_n in eine Bytefolge bzw. einen Bytestrom von kürzerer Länge, wobei diese von der gewünschten Kompressionsrate abhängt. Wenn zum Beispiel N=160 und eine sehr hohe Datenkompression gewünscht ist, kann der Ausgangsbytestrom bis zu 12 Acht-Bit-Bytes kurz sein. Ein Blockdiagramm der Codiereinrichtung ist in Fig. 4 dargestellt.

Demzufolge beginnt die Routine mittels Annehmens eines Rahmens s_n (Block 50). Um Niederfrequenzrauschen zu filtern bzw. zu entfernen, wie zum Beispiel Wechselstrom oder 60 Hz Stromleitungsrauschen, und zum Erzeugen offsetfreier Sprachdaten, wird das Signal s_n durch einen Hochpassfilter geführt. Eine Differenzengleichung, die in einem bevorzugten System zur Erzielung dieser Aufgabe verwendet wird, ist in Gleichung 1 angegeben für $0 \leq n < N$.

$$x_n = s_n - s_{n-1} + 0.999 * x_{n-1} \qquad \text{Gleichung 1}$$

Der Wert X_n ist das „offsetfreie" Signal. Die Variablen s_{-1} und x_{-1} sind für jeden Diphon auf Null initialisiert und werden anschließend aktualisiert unter Verwendung der Beziehung von Gleichung 2.

$$x_{-1} = x_N \text{ und } s_{-1} = s_N \qquad \text{Gleichung 2}$$

Dieser Schritt kann als Offsetkompensierung oder Wechselstromentfernen bezeichnet werden (Block 51).

Um ein teilweises Entkorrelieren der Sprachproben und des Quantisierungsrauschens zu erreichen, wird die Sequenz x_n durch einen festgelegten bzw. festen Erstordnungslinearprädiktionsfilter geführt. Die Differenzengleichung, um dies zu erreichen, ist in Gleichung 3 angegeben.

$$y_n = x_n - 0.875 * x_{n-1} \qquad \text{Gleichung 3}$$

Die lineare Prädiktionsfilterung gemäß Gleichung 3 erzeugt einen Rahmen y_n (Block 52). Der Filterparameter, welcher in Gleichung 3 gleich 0.875 ist, muß verändert werden, wenn eine andere Sprachsample- bzw. probennahmenrate verwendet

wird. Der Wert von x_{-1} ist für jeden Diphon auf Null initialisiert, wird jedoch in dem Schritt der umgekehrten Linearprädiktionsfilterung (Block 60) aktualisiert, wie es folgend beschrieben wird.

Es ist möglich, eine Vielzahl von Filtertypen zu verwenden, einschließlich zum Beispiel eines adaptiven bzw. adaptativen Filters, in welchem die Filterparameter von den zu kodierenden Diphonen abhängen, oder auch Filter von höherer Ordnung.

Die Sequenz y_n, die durch die Gleichung 3 erzeugt wird, wird anschließend verwendet, um einen optimalen Pitchwert P_{opt} zu bestimmen, sowie einen zugeordneten Verstärkungsfaktor β. P_{opt} wird berechnet unter Verwendung der Funktionen $s_{xy}(P)$, $s_{xx}(P)$, $s_{yy}(P)$ sowie der Kohärenzfunktion Coh(P), die durch die Gleichungen 4, 5, 6 und 7 definiert ist, wie folgend dargelegt.

$$S_{xy}(P)=\sum_{n=0}^{N-1} y_n * PBUF_{P_{max}-P+n} \qquad \text{Gleichung 4}$$

$$S_{xx}(P)=\sum_{n=0}^{N-1} y_n * y_n \qquad \text{Gleichung 5}$$

$$S_{yy}(P)=\sum_{n=0}^{N-1} PBUF_{P_{max}-P+n} * PBUF_{P_{max}-P+n} \qquad \text{Gleichung 6}$$

und

$$Coh(P) = s_{xy}(P) * s_{xy}(P) / (s_{xx}(P) * s_{yy}(P)) \qquad \text{Gleichung 7}$$

PBUF ist ein Pitchpuffer der Größe P_{max}, welcher auf Null initialisiert ist und in dem Aufteilungspuffer-Aktualisierblock 59 aktualisiert wird wie folgend beschrieben. P_{opt} ist der Wert von P, für welchen Coh(P) maximal und $s_{xy}(P)$ positiv ist. Der betrachtete Bereich von P ist abhängig von dem Nominalpitch der zu codierenden Sprache. Der Bereich liegt zwischen 96 und 350, wenn die Rahmengröße gleich 96 ist, und liegt zwischen 160 und 414, wenn die Rahmengröße gleich 160 ist. P_{max} ist 350, wenn der Nominalpitch geringer als 160 ist und ist ansonsten gleich 414. Der Parameter P_{opt} kann unter Verwendung von acht Bits dargestellt werden.

Die Berechnung von P_{opt} kann verstanden werden unter Bezugnahme auf Fig. 5. In Fig. 5 ist der Puffer PBUF dargestellt durch die Sequenz 100, wobei der Rahmen y_n dargestellt wird durch die Sequenz 101. In einem Segment von Sprachdaten, in welchem die vorangegangen Rahmen im wesentlichen gleich sind zu dem Rahmen y_n, werden PBUF und y_n aussehen, wie es in Fig. 5 gezeigt ist. P_{opt} wird den Wert an dem Punkt 102 annehmen, an welchem der Vektor y_n 101 so weit wie möglich an ein entsprechendes Segment von ähnlicher Länge in dem PBUF 100 angepaßt ist.

Der Pitchfilterverstärkungsparameter β wird bestimmt unter der Verwendung des Ausdruckes von Gleichung 8.

$$\beta = s_{xy}(P_{opt}) \;/\; s_{yy}(P_{opt}) \qquad \text{Gleichung 8}$$

β ist diskretisiert bzw. quantisiert auf vier Bits, so daß der quantisierte Wert von β in einem Bereich liegen kann von 1/16 bis 1, und zwar bei Schritten von 1/16.

Anschließend wird ein Pitchfilter angewendet (Block 54). Die Langzeitkorrelationen in den vorverstärkten bzw. Vorbetonten Sprachdaten yy_n werden entfernt unter Verwendung der Beziehung von Gleichung 9.

$$r_n = y_n - \beta * PBUF_{P_{max}-P_{opt}+n} \qquad 0 \quad n < N.$$

Gleichung 9

Dies führt zu einer Berechnung eines Restsignales r_n.

Anschließend wird ein Skalierungsparameter G generiert unter Verwendung einer Blockverstärkungsschätzroutine (Block 55). Um die Berechnungsgenauigkeit der folgenden Verarbeitungsstufen zu erhöhen, wird das Restsignal r_n nachskaliert. Der Skalierungsparameter G wird erhalten, indem zuerst der größte Betrag bzw. Ausschlag des Signales r_n bestimmt und quantisiert wird, unter Verwendung einer 7-Pegelquantisiereinrichtung. Der Parameter G kann einen der folgenden sieben Werte annehmen: 256, 512, 1024, 2048, 4096, 8192 und 16384. Die Konsequenz des Auswählens dieser Quantisierungspegel ist, daß das Nachskalierverfahren lediglich unter Verwendung von Verschiebeschritten implementiert werden kann.

Anschließend schreitet die Routine fort zu der verbleibenden Codierung unter Verwendung eines Vollsuchvektorquantisiercodes (Block 56). Um das Restsignal r_n zu codieren, wird die n-Punkt-Sequenz r_n aufgeteilt in nichtüberlappende Blöcke der Länge M, wobei M als „Vektorgröße" bez ichnet wird. Demzufolge werden M Probenblöcke b_{ij} erzeugt, wobei i ein Index von Null bis M-1 bezüglich der Blockzahl ist, und wobei j ein Index von Null bis N/M-1 bezüglich der Probe

innerhalb des Blockes ist. Jeder Block kann in der Weise definiert werden, wie in Gleichung 10 angegeben.

$$b_{ij} = r_{Mi+j}, \quad (0 \leq i < N/M \text{ und } j \leq 0 < M)$$

Gleichung 10

Jeder dieser M Probenblöcke b_{ij} wird in eine Acht-Bit-Zahl unter Verwendung von Vektorquantisierung codiert. Der Wert von M ist abhängig von dem gewünschten Kompressionsverhältnis. Wenn zum Beispiel M gleich 16 ist, wird eine sehr hohe Kompression erzielt (d.h. 16 Restproben werden unter Verwendung von lediglich acht Bits codiert). Die decodierte Sprachqualität kann jedoch als relativ Geräusch- bzw. Rauschbelastet wahrgenommen werden, wenn M = 16 ist. Andererseits wird mit M = 2 die dekomprimierte Sprachqualität sehr nahe bei jener von nichtkomprimierter Sprache vorliegen. Die Länge der komprimierten Sprachaufzeichnungen wird jedoch länger sein. Bei der bevorzugten Implementierung kann der Wert M Werte annehmen von 2, 4, 8 und 16.

Die Vektorquantisierung wird in solch einer Weise durchgeführt, wie es in Fig. 6 dargestellt ist. Demzufolge wird für sämtliche Blöcke b_{ij} eine Sequenz von Quantisierungsvektoren identifiziert (Block 120). Anfänglich werden die Komponenten des Blockes b_{ij} durch einen Rauschformfilter geführt und gemäß der Gleichung 11 skaliert (Block 121).

$$w_j = 0.875 * w_{j-1} - 0.5 * w_{j-2} + 0.4375 * w_{j-3} + b_{ij}, \quad 0 \leq j < M; \quad v_{ij} = G * w_j;$$
$$0 \leq j < M$$

Gleichung 11

Somit ist v_{ij} die j-te Komponente des Vektors v_i, wobei die Werte w_{-1}, w_{-2} und w_{-3} die Zustände des Geräusch- bzw.

Rauschformfilters sind, wobei diese für jedes Diphon auf Null initialisiert sind. Die Filterkoeffizienten sind ausgewählt zum Formen des Quantisierrauschspektrums, um die subjektive Qualität der dekomprimierten Sprache zu verbessern. Nachdem jeder Vektor codiert und entcodiert ist, werden diese Zustände aktualisiert, wie folgend beschrieben unter Bezugnahme auf die Blöcke 124 bis 126.

Anschließend findet die Routine Zeiger zu der besten Übereinstimmung in einer Vektorquantisiertabelle (Block 122). Die Vektorquantisiertabelle 123 besteht aus einer Sequenz von Vektoren C_0 bis C_{255} (Block 123).

Demzufolge wird der Vektor v_i verglichen mit 256 M-Punktvektoren, welche vorberechnet und in der Codetabelle 123 gespeichert sind. Der Vektor C_{qi}, welcher am nähesten zu v_i ist, wird entsprechend der Gleichung 12 bestimmt. Der Wert C_p für p = 0 bis 255 repräsentiert den p-ten Codiervektor aus der Vektorquantisiercodetabelle 123.

$$\min_{p} \sum_{j=0}^{M-1} (v_{ij} - C_{pj})^2$$

Gleichung 12

Der nächste bzw. vergleichbarste Vektor C_{qi} kann ebenfalls effizient bestimmt werden unter Verwendung der Technik nach Gleichung 13.

$$v_i^T \bullet C_{qi} \leq v_i^T \bullet C_p \text{ für sämtliche } p(0 \leq p \leq 255)$$

Gleichung 13

In Gleichung 13 repräsentiert der Wert v^T die Transponierte des Vektors v, wobei „•" die innere Produktoperation in der Ungleichung wiedergibt.

Die Codiervektoren C_p in Tabelle 123 werden verwendet, um angepaßt zu sein bezüglich des Rauschen-gefilterten Werts v_{ij}. Beim Decodieren wird jedoch eine Decodiervektortabelle 125 verwendet, welche aus einer Sequenz von Vektoren QV_p besteht. Die Werte QV_p sind zu dem Zweck ausgewählt, daß qualitativ hochwertig Ton- bzw. Lautdaten erreicht werden unter Verwendung der Vektorquantisiertechnik. Demzufolge wird nach dem Auffinden des Vektors C_{qi} der Zeiger q verwendet, um auf den Vektor QV_{qi} zuzugreifen. Die decodierten Proben, die dem Vektor b_i entsprechen, der bei Schritt 55 von Fig. 4 erzeugt wurde, ist der M-Punktvektor (1/G) * QV_{qi}. Der Vektor C_p steht mit dem Vektor QV_p über den Rauschformfilterbetrieb nach Gleichung 11 in Verbindung. Wenn demzufolge auf den Decodiervektor QV_p zugegriffen wird, so muß kein Umkehr- bzw. Inversrauschenformfilter in dem Decodierverfahren berechnet werden. Die Tabelle 125 von Fig. 6 enthält somit rauschkompensierte Quantisierungsvektoren.

Bei der Weiterführung der Berechnung der Codiervektoren für die Vektoren b_{ij}, welche das Restsignal r_n bilden, wird auf den Decodiervektor des Zeigers zu dem Vektor b_i zugegriffen (Block 124). Dieser Decodiervektor wird zur Filterung und PBUF-Aktualisierung verwendet (Block 126).

Für den Rauschformfilter wird, nachdem die decodierten Proben für jeden Unterblock b_i berechnet sind, der Fehlervektor (b_i-QV_{qi}) durch den Rauschformfilter geführt, wie es in Gleichung 14 dargestellt ist.

$$W_j = 0.875 * W_{j-1} - 0.5 * W_{j-2} + 0.4375 * W_{j-3} + [b_{ij} - QV_{qi}(j)];$$

$$0 \leq j < M$$

Gleichung 14

In Gleichung 14 repräsentiert der Wert $QV_{qi}(j)$ die j-te Komponente des Decodiervektors QV_{qi}. Die Rauschformfilterzustände für den nächsten Block werden wie in Gleichung 15 gezeigt aktualisiert.

$$w_{-1} = w_{M-1}$$
$$w_{-2} = w_{M-2}$$
$$w_{-3} = w_{M-3}$$

Gleichung 15

Dieses Codieren und Decodieren wird für sämtliche der N/M Unterblöcke durchgeführt, um N/M Indizes auf die Decodiervektortabelle 125 zu erhalten. Dieser String an Indizes Q_n, wobei n Werte zwischen Null und N/M-1 annimmt, repräsentiert Identifikationen für einen String von Decodiervektoren für das Restsignal r_n.

Somit repräsentieren vier Parameter die N-Punkt-Datensequenz y_n:

1. optimaler Pitch, P_{opt} (8 Bits),
2. Pitchfilterverstärkung, β (4 Bits),
3. Skalierparameter G (3 Bits), und
4. ein String von Decodiertabellenindizes Q_n ($0 \leq n < N/M$).

Die Parameter β und G können zu einem einzelnen Byte codiert werden. Somit werden lediglich (N/M) plus 2 Bytes zum Repräsentieren von N Sprachproben verwendet. Es wird beispielhaft angenommen, daß der Nominal- bzw. Nennpitch 100

Proben lang ist, wobei M=16 gilt. In diesem Fall wird ein Rahmen von 96 Sprachproben wiedergegeben durch 8 Bytes: 1 Byte für P_{opt}, 1 Byte für β und G und 6 Bytes für die Decodiertabellenindizes Q_n. Wenn die nichtkomprimierte Sprache aus 16-Bit-Proben besteht, so stellt dies eine Kompression von 24:1 dar.

Unter erneuter Bezugnahme auf Fig. 4 werden vier Parameter, die die Sprachdaten identifizieren, gespeichert (Block 57). Bei einem bevorzugten System werden diese in einer Struktur gespeichert, wie es unter Bezugnahme auf Fig. 3 beschrieben wurde, wobei die Struktur des Rahmens wie folgt gekennzeichnet werden kann:

```
    #define   NumOfVectorsPerFrame     (FrameSize / VectorSize)

    struct frame {
        unsigned  Gain : 4;
        unsigned  Beta : 3;
        unsigned  UnusedBit: 1;
        unsigned  char Pitch ;
        unsigned  char Vqcodes [NumOfVectorsPerFrame]; };
```

Die Diphonaufzeichnung von Fig. 3, die diese Rahmenstruktur verwendet, kann wie folgt gekennzeichnet werden:

```
    Diphonaufzeichnung
    {
        char LeftPhone, RightPhone;
        short     LeftPitchPeriodCount,     RightPitchPeriodCount;
```

```
    short       *LeftPeriods, *RightPeriods;
    struct      frame *LeftData, *RightData;
}
```

Die gespeicherten Parameter stellen eine einzigartige Identifikation der Diphone bereit, die für die Text-zu-Sprach-Synthese erforderlich sind.

Wie oben angegeben unter Bezugnahme auf Fig. 6, führt die Codiereinrichtung das Decodieren der Daten fort, die codiert sind bzw. werden, um die Filter- und PBUF- Werte zu aktualisieren. Der erste hierbei durchgeführte Schritt ist ein Invers- bzw. Umkehrpitchfilter (Block 58). Wenn der Vektor r_n, der dem decodierten Signal entspricht, welches durch die Verkettung des Strings von Decodiervektoren gebildet ist, um das Restsignal r_n darzustellen, wird der Inversfilter implementiert wie in Gleichung 16 dargestellt.

$$y_n = r_n + \beta * PBUF_{Pmax - Popt + n}; \quad 0 \leq n < N$$

Gleichung 16

Anschließend wird der Pitchpuffer aktualisiert (Block 59), und zwar mit der Ausgabe des Umkehrpitchfilters. Der Aufteilungspuffer PBUF wird aktualisiert, wie es in Gleichung 17 angegeben ist.

$$PBUF_n = PBUF_{(n+N)}; \qquad 0 \leq n < (P_{max} - N)$$
$$PBUF_{(Pmax - N + n)} = y_n; \quad 0 \leq n < N \qquad \text{Gleichung 17}$$

Schließlich werden die Linearprädiktionsfilterparameter unter Verwendung eines Inverslinearprädiktionsfilterschrittes (Block 60) aktualisiert. Das Ausgangssignal bzw. die Ausgabe des Inverspitchfilters wird durch einen Umkehrlinearprä-

diktionsfilter erster Ordnung geführt, um die decodierte Sprache zu erhalten. Die Differenzgleichung zum Implementieren dieses Filters ist in Gleichung 18 angegeben.

$$x_n = 0.875 * x_{n-1} + Y_n$$

Gleichung 18

In Gleichung 18 ist x_n die dekomprimierte Sprache. Hieraus wird der Wert von x_{-1} für den nächsten Rahmen zur Verwendung in dem Schritt des Blockes 52 eingestellt auf den Wert x_M.

Fig. 7 stellt die Decodierroutine dar. Das Decodermodul akzeptiert als Eingabe bzw. Eingangssignal (N/M) + 2 Bytes an Daten, welche durch das Codiermodul erzeugt sind und wendet als Ausgabe N Sprachproben an. Der Wert von N wird abhängig von dem Nennpitch der Sprachdaten sein, wobei der Wert von M abhängig ist von dem gewünschten Kompressionsverhältnis.

Bei ausschließlich softwarebasierenden Text-zu-Sprach-Systemen muß die Berechnungskomplexität des Decodierers so gering wie möglich sein, um sicherzustellen, daß das Text-zu-Sprach-System in Echtzeit laufen kann, und zwar selbst auf langsamen Computern. Ein Blockdiagramm des Decodierers ist in Fig. 7 gezeigt.

Die Routine beginnt durch die Annahme von Diphonaufzeichnungen bei Block 200. Der erste Schritt beinhaltet das Abschätzen der Parameter G, β, P_{opt} und des Vektorquantisierstrings Q_n (Block 201). Anschließend wird das Restsignal r_n decodiert (Block 202). Dies umfaßt das Zugreifen auf und das Verketten der Decodiervektoren für den Vektorquantisierstring, wie es schematisch beim Block 203 gezeigt ist, mit einem Zugriff auf die Decodiervektortabelle 125.

Nachdem das Restsignal r_n decodiert ist, wird ein Umkehrpitchfilter angewendet (Block 204). Dieser Umkehr- bzw. Inverspitchfilter wird implementiert, wie es in Gleichung 19 gezeigt ist:

$$y_n = r_n + \beta * SPBUF\ (P_{max} - P_{opt} + n),\ 0 \leq n < N.$$

Gleichung 19

SPBUF ist ein Synthetisierpitch- bzw. -Aufteilungspuffer der Länge P_{max}, und zwar initialisiert auf Null für jeden Diphon, wie oben beschrieben, unter Bezugnahme auf den Codierpitchpuffer PBUF.

Für jeden Rahmen wird der Synthesepitchpuffer aktualisiert (Bock 205). Die Weise, in welcher dieses Aktualisieren erfolgt, ist in Gleichung 20 dargestellt:

$$SPBUF_n = SPBUF_{(n+N)} \qquad 0 \leq n < (P_{max} - N)$$
$$SPBUF_{(Pmax - N + n)} = y_n \qquad 0 \leq n < N$$

Gleichung 20

Nach dem Aktualisieren von SPBUF wird die Sequenz y_n auf den Inverslinearprädiktionsfilterschritt angewendet (Block 206). Somit wird die Ausgabe des Umkehrpitchfilters y_n durch einen Inverslinearprädiktionsfilter erster Ordnung geführt, um die decodierte Sprache zu erhalten. Die Differenzgleichung zum Implementieren des Umkehrlinearprädiktionsfilters ist in Gleichung 21 angegeben:

$$x_n = 0.875 * x_{n-1} + y_n$$

Gleichung 21

In Gleichung 21 entspricht der Vektor x_n der dekomprimierten Sprache. Dieses Filterverfahren kann implementiert werden unter Verwendung einfacher Verschiebeschritte bzw. -Handhabungen und erfordert keine Multiplikation. Demzufolge kann eine schnelle Ausführung gewährleistet werden, wobei sehr geringe Mengen an Hostcomputerressourcen verwendet werden.

Codieren und Decodieren von Sprache gemäß den oben beschriebenen Algorithmen bietet mehrere Vorteile gegenüber Systemen gemäß dem Stand der Technik. In erster Linie bietet diese Technik höhere Sprachkompressionsraten mit Decodierern, die einfach genug sind, um in der Implementierung von reinen Software-Text-zu-Sprach-Systemen an Computersystemen mit niedriger Verarbeitungsleistung verwendet zu werden. Zum anderen bietet die Technik ein sehr flexibles Verhältnis zwischen dem Kompressionsverhältnis und der Synthetisiersprachqualität. Ein Hochleistungscomputersystem kann für eine synthetisierte Sprache von höherer Qualität zu Lasten einer stärkeren RAM-Speicheranforderung ausgelegt werden.

III. Wellenformvermischung für Diskontinuitätenglättung (Fig. 8 und 9)

Wie oben unter Bezugnahme auf Fig. 2 erwähnt, können die synthetisierten Rahmen von Sprachdaten, erzeugt unter Verwendung der Vektorquantisiertechnik, zu leichten Diskontinuitäten zwischen Diphonen in einem Textstring führen. Demzufolge stellt das Text-zu-Sprach-System ein Modul bereit zum Vermischen der Diphondatenrahmen, um solche Diskontinuitäten zu glätten. Die Vermischungs- bzw. Abmischtechnik

gemäß der bevorzugten Ausführungsform wird dargestellt unter Bezugnahme auf die Fig. 8 und 9.

Zwei verkettete Diphone werden einen Endungsrahmen und einen Anfangsrahmen aufweisen. Der Endungsrahmen des linken Diphons muß vermischt werden mit dem Anfangsrahmen des rechten Diphons, und zwar ohne daß hörbare Diskontinuitäten oder Klicks erzeugt würden. Da der rechte Rand des ersten Diphons und der linke Rand des zweiten Diphons in den meisten Situationen demselben Phonem entsprechen, wird von ihnen angenommen, daß sie eine ähnliche Erscheinung an dem Verkettungspunkt aufweisen. Da jedoch die zwei Diphoncodierungen aus einem unterschiedlichen Kontext extrahiert werden, werden sie nicht identisch zueinander sein. Diese Vermischungstechnik wird angewendet zum Eliminieren von Diskontinuitäten an dem Verkettungspunkt. In Fig. 9 ist der letzte Rahmen, Bezugnahme wird hier auf eine Pitchperiode genommen, des linken Diphons mit L_n beziffert ($0 \leq n < PL$), und zwar am oberen Rand der Seite. Der erste Rahmen (Pitchperiode) des rechten Diphons ist mit R_n beziffert ($0 \leq n < PR$). Das Vermischen von L_n und R_n gemäß der vorliegenden Erfindung wird lediglich diese zwei Pitchperioden verändern und wird durchgeführt wie beschrieben unter Bezugnahme auf Fig. 8. Die Wellenformen in Fig. 9 sind gewählt zur Darstellung des Algorithmus und müssen nicht repräsentativ für reale Sprachdaten sein.

Demzufolge beginnt der Algorithmus, wie in Fig. 8 gezeigt, mit dem Empfang des linken und rechten Diphons in einer Sequenz (Block 300). Anschließend wird der letzte Rahmen des linken Diphons in dem Puffer L_n gespeichert (Block 301). Ferner wird der erste Rahmen des rechten Diphons in dem Puffer R_n (Block 302) gespeichert.

Anschließend repliziert der Algorithmus und verkettet den linken Rahmen L_n um einen erweiterten bzw. ausgeweiteten Rahmen zu bilden (Block 303). In dem anschließenden Schritt werden die Diskontinuitäten in dem erweiterten Rahmen zwischen den replizierten linken Rahmen geglättet (Block 304). Dieser geglättete und erweiterte linke Rahmen wird als EI_n in Fig. 9 bezeichnet.

Die erweiterte Sequenz EI_n ($0 \leq n < 2*PL$) wird in dem ersten Schritt erhalten, wie in Gleichung 22 gezeigt:

$$EI_n = L_n \qquad n = 0,1,\ldots, PL-1$$
$$EI_{PI+n} = L_n \qquad n = 0,1,\ldots, PL-1$$

Gleichung 22

Anschließend wird die Diskontinuitätenglättung von dem Punkt n = PL ausgeführt, und zwar entsprechend dem Filter von Gleichung 23:

$$EI_{PI+n} = EI_{PI+n} + [EI_{(PI-1)} - EI_{(PI-1)}] * \Delta^{n+1},$$

$$n = 0,1,\ldots, (PL/2).$$

Gleichung 23

In Gleichung 23 ist der Wert Δ gleich 15/16, wobei $EI_{(PI-1)} = EI_2 + 3 * (EI_1 - EI_0)$. Demzufolge, und wie es in Fig. 9 angegeben ist, ist die erweiterte Sequenz EI_n im wesentlichen gleich L_n an der linken Seite und verfügt über einen geglätteten Bereich, beginnend an dem Punkt PL und übergehend zu der Original- bzw. Ursprungsform von L_n hin zu dem Punkt 2PL. Wenn L_n perfekt periodisch war, so gilt $EI_{PL-1} = EI_{PL-1}$.

In dem anschließenden Schritt wird die optimale Abstimmung bzw. Anpassung von R_n mit dem Vektor EI_n ermittelt. Dieser Übereinstimmungspunkt wird als P_{opt} bezeichnet (Block 305). Dies wird im wesentlichen erzielt, wie es in Fig. 9 gezeigt ist, durch Vergleichen von R_n mit EI_n, um den Abschnitt von EI_n zu finden, welcher am besten zu R_n paßt bzw. am deutlichsten mit diesem übereinstimmt. Diese optimale Vermischungspunktbestimmung wird durchgeführt unter Verwendung von Gleichung 24, wobei W das Minimum von PL und PR ist, und wobei AMDF die mittlere Betragsdifferenzfunktion wiedergibt.

$$AMDF(p) = \sum_{n=0}^{W-1} | EI_{n+p} - R_n |$$

Gleichung 24

Diese Funktion wird für Werte von p in dem Bereich von 0 bis PL-1 berechnet. Die vertikalen Striche in der Operation bezeichnen den Absolutwert. W ist die Fenstergröße für die AMDF Berechnung. P_{opt} wird als Wert gewählt, bei welchem AMDF(p) minimal ist. Dies bedeutet, daß $p = P_{opt}$ dem Punkt entspricht, an welchem die Sequenzen $EI_{n+p}(0 \leq n < W)$ und $R_n(0 \leq n < W)$ sehr nah beieinander liegen.

Nach der Bestimmung des optimalen Vermischungspunktes P_{opt}, werden die Wellenformen vermischt (Block 306). Das Vermischen verwendet eine erste Gewichtungsrampe bzw. -funktion WL, welche in Fig. 9 gezeigt ist als bei P_{opt} in der EI_n-Spur beginnend. In einer zweiten Rampe ist WR in Fig. 9 gezeigt bei der R_n-Spur, welche aufgezeichnet ist mit P_{opt}. Somit wird beim Anfang des Vermischungsverfahrens der Wert

von EI_n hervorgehoben bzw. betont. An dem Ende des Vermischungsverfahrens wird der Wert von R_n hervorgehoben.

Vor dem Vermischen bzw. Abmischen wird die Länge PL von L_n nach Bedarf verändert, um zu sichern, daß, wenn die modifizierten L_n und R_n verkettet werden, die Wellenformen möglichst kontinuierlich sind. Somit wird die Länge P L eingestellt auf P_{opt}, wenn P_{opt} größer als PL/2 ist. Ansonsten ist die Länge P L gleich W + P_{opt}, wobei die Sequenz L_n gleich ist EI_n für $0 \leq n \leq (P\ L-1)$.

Die Vermischungsrampe, die beim Punkt P_{opt} beginnt, ist in Gleichung 25 angegeben:

$$R_n = EI_{n+Popt} + (R_n - EI_{n+Popt}) * (n+1)/W \qquad 0 \leq n < W$$

$$R_n = R_n \qquad W \leq n < PR$$

Gleichung 25

Die Sequenzen L_n und R_n werden somit fenstermäßig verwaltet und addiert, um den bzw. die abgemischten R_n zu erhalten. Der Anfang von L_n und die Endung von R_n werden beibehalten, um jegliche Diskontinuitäten mit benachbarten Rahmen zu vermeiden.

Diese Vermischungstechnik wird erachtet als Vermischungsrauschen minimierend bei synthetisierter Sprache, die durch beliebige Verkettungssprachsynthese erzeugt ist.

IV. Pitch- und Dauermodifikation (Fig. 10 bis 18)

Wie oben unter Bezugnahme auf Fig. 2 angegeben, untersucht ein Textanalysierprogramm den Text und bestimmt die Dauer und die Pitchkontour von jedem Phon, welches synthetisiert

werden muß, und erzeugt Betonungssteuersignale. Eine typische Steuerung für ein Phon wird angeben, daß ein gegebenes Phonem, wie zum Beispiel AE, eine Dauer von 200 Millisekunden aufweisen sollte, wobei ein Pitch linear von 220 Hz auf 300 Hz ansteigen sollte. Diese Anforderung ist graphisch in Fig. 10 gezeigt. Wie es in Fig. 10 gezeigt ist, ist T gleich der gewünschten Dauer (z.B. 200 Millisekunden) des Phonems. Die Frequenz f_b ist der gewünschte Anfangspitch in Hz. Die Frequenz f_e ist der gewünschte Endpitch in Hz. Die Markierungen bzw. Label P_1, P_2, ...,P_6 geben die Zahl von Proben in jedem Rahmen an, um die gewünschten Pitchfrequenzen f_b, f_2..., f_6 zu erhalten. Die Beziehung zwischen der gewünschten Zahl von Proben, P_i, und der gewünschten Pitchfrequenz f_i ($f_1 = f_b$) wird definiert durch die Beziehung:

$P_i = F_s/f_i$, wobei F_s die Abtastfrequenz für die Daten ist.

Wie es in Fig. 10 zu sehen ist, ist die Pitchperiode für eine Niederfrequenzperiode des Phonems länger als die Pitchperiode für eine höherfrequente Periode des Phonems. Wenn die Nominalfrequenz P_3 wäre, so würde der Algorithmus die Pitchperiode für die Rahmen P_1 und P_2 verlängern, und die Pitchperioden für die Rahmen P_4, P_5 und P_6 absenken müssen. Auch die gegebene Dauer T des Phonems wird angeben, wie viele Aufteilungs- bzw. Pitchperioden eingeführt oder von dem codierten Phonem gelöscht werden sollten, um die gewünschte Periodendauer zu erhalten. Fig. 11 bis 18 stellen eine bevorzugte Implementierung solcher Algorithmen dar.

Fig. 11 stellt einen Algorithmus dar zur Vergrößerung der Pitchperiode, und zwar unter Bezugnahme auf die Kurven bzw.

Graphen von Fig. 12. Der Algorithmus beginnt, indem eine Steuerung empfangen wird zum Vergrößern der Pitchperiode auf $N + \Delta$, wobei N die Pitch- periode des codierten Rahmens ist (Block 350). In dem nächsten Schritt werden die Pitchperiodendaten in einem Puffer x_n gespeichert (Block 351). x_n ist in Fig. 12 in dem oberen Abschnitt der Seite dargestellt. In dem anschließenden Schritt wird ein Linksvektor bzw. ein linker Vektor L_n generiert durch Anwenden einer Gewichtungsfunktion WL auf die Pitchperiodendaten x_n, und zwar unter Bezugnahme auf Δ (Block 352). Diese Gewichtungsfunktion ist in Gleichung 26 angegeben, wobei gilt $M = N - \Delta$:

$$L_n = x_n \qquad \text{für } 0 \leq n < \Delta$$

$$L_n = x_n * (N-n) / (M+1) \qquad \text{für } \Delta \leq n < N$$

Gleichung 26

Wie es in Fig. 12 zu sehen ist, ist die Gewichtungsfunktion WL konstant von der ersten Probe zu der Probe Δ und nimmt von Δ zu N ab.

Anschließend wird eine Gewichtungsfunktion WR auf x_n angewendet (Block 353), wie es in Fig. 12 zu sehen ist. Diese Gewichtungsfunktion wird ausgeführt, wie es in Gleichung 27 gezeigt ist:

$$R_n = x_{n+\Delta} * (n + 1) / (M + 1) \qquad \text{für } 0 \leq n < N - \Delta$$

$$R_n = x_{n+\Delta} \qquad \text{für } N - \Delta \leq n < N$$

Gleichung 27

Wie es in Fig. 12 zu sehen ist, nimmt die Gewichtungsfunktion WR von 0 zu $N - \Delta$ zu und verbleibt konstant von $N - \Delta$ bis hin zu N. Die resultierenden Wellenformen L_n und R_n

sind konzeptartig in Fig. 12 dargestellt. Wie zu erkennen ist, behält L_n den Anfang der Sequenz x_n bei, während R_n das Ende bzw. die Endung der Daten x_n beibehält.

Die pitchmodifizierte Sequenz y_n wird gebildet (Block 354), indem die zwei Sequenzen addiert werden, wie es in Gleichung 28 gezeigt ist:

$$y_n = L_n + R_{(n - \Delta)} \qquad \text{Gleichung 28}$$

Dies ist graphisch in Fig. 12 gezeigt, indem R_n um Δ verschoben unterhalb von L_n angeordnet ist. Die Kombination von L_n und R_n, verschoben um Δ, ist als y_n am unteren Rand von Fig. 12 gezeigt. Die Pitchperiode für y_n ist $N + \Delta$. Der Anfang von y_n ist entsprechend bzw. identisch zu dem Anfang von x_n, wobei die Endung von y_n im wesentlichen identisch bzw. entsprechend ist zu der Endung von x_n. Somit kann eine Kontinuität mit benachbarten Rahmen in der Sequenz beibehalten werden, wobei ein glatter Übergang erzielt wird, während die Pitchperiode der Daten erweitert wird.

Gleichung 28 wird ausgeführt unter der Annahme, daß L_n Null ist, und zwar für $n \leq N$, und R_n Null ist, und zwar für $n<0$. Dies ist bildartig in Fig. 12 dargestellt.

Eine effiziente Implementierung dieses Schemas, welches höchstens eine Multiplikation je Probe erfordert, ist in Gleichung 29 gezeigt:

$$y_n = x_n \qquad 0 \leq n < \Delta$$

$$y_n = x_n + [x_{n-\Delta} - x_n] * (n - \Delta + 1) / (N - \Delta + 1)$$

$$\Delta \leq n < N$$

$Y_n = x_{n-\Delta}$ $\quad N \leq n < N_d$

Gleichung 29

Dies resultiert in einer neuen Pitchperiode mit einer Pitchperiode von $N + \Delta$.

Es kann ebenfalls vorkommen, daß die Pitchperiode verkleinert werden muß. Der Algorithmus zum Absenken bzw. Verkleinern der Pitchperiode ist in Fig. 13 gezeigt unter Bezugnahme auf die Graphen von Fig. 14. Demzufolge beginnt der Algorithmus mit einem Steuersignal, welches angibt, daß die Pitchperiode verringert werden muß auf $N-\Delta$ (Block 400). Der erste Schritt besteht darin, zwei aufeinanderfolgende Pitchperioden in dem Puffer x_n zu speichern (Block 401). Somit besteht der Puffer x_n, wie es in Fig. 14 zu sehen ist, aus zwei aufeinanderfolgenden Pitchperioden, wobei die Periode N_1 die Länge der ersten Pitchperiode und N_r die Länge der zweiten Pitchperiode ist. Anschließend werden zwei Sequenzen L_n und R_n konzeptartig erzeugt, unter Verwendung von Gewichtungsfunktionen WL und WR (Blöcke 402 und 403). Die Gewichtungsfunktion WL betont den Anfang der ersten Pitchperiode, wobei die Gewichtungsfunktion WR die Endung der zweiten Pitchperiode betont. Diese Funktionen können konzeptartig wiedergegeben werden, wie es in den Gleichungen 30 bzw. 31 gezeigt ist:

$L_n = x_n$ $\quad$ für $0 \leq n < N_1 - W$

$L_n = x_n * (N_1-n)/(W+1)$ $\quad W \leq n < N_1$

$L_n = 0$ $\quad$ für andere Fälle

Gleichung 30

$R_n = x_n * (n-N_1+W-\Delta+1)/(W+1)$ $\quad$ für $N_1-W+\Delta \leq n < N_1+\Delta$

$R_n = x_n$ $\quad$ für $N_1+\Delta \leq n < N_1+N_r$

$R_n = 0$ für alle anderen Fälle

Gleichung 31

In diesen Gleichungen ist Δ gleich der Differenz zwischen N_1 und der gewünschten Pitchperiode N_d. Der Wert W ist gleich zu 2 * Δ, insoweit 2 * Δ nicht größer als N_d ist, in welchem Fall W gleich N_d ist.

Diese zwei Sequenzen L_n und R_n werden vermischt zur Bildung einer pitchmodifizierten Sequenz y_n (Block 404). Die Länge der pitchmodifizierten Sequenz y_n wird gleich sein der Summe der gewünschten Länge der Länge des rechten Phonemrahmens N_r. Sie wird gebildet durch Addieren der zwei Sequenzen, wie es in Gleichung 32 gezeigt ist:

$$y_n = L_n + R_{(n+\Delta)}$$ Gleichung 32

Demzufolge werden zwei aufeinanderfolgende Pitchperioden von Daten beeinflußt, wenn eine Pitchperiode verkleinert wird, obwohl lediglich die Länge von einer Pitchperiode verändert wird. Dies erfolgt, da Pitchperioden an Orten aufgeteilt sind bzw. werden, an welchen Kurzzeitenergie am niedrigsten innerhalb einer Pitchperiode ist. Somit beeinflußt diese Strategie lediglich den Niederenergieabschnitt der Pitchperioden. Dies minimiert die durch die Pitchmodifikation bedingte Beeinträchtigung der Sprachqualität. Es sei zu verstehen gegeben, daß die Zeichnungen in Fig. 14 vereinfacht sind und keine tatsächlichen Pitchperiodendaten wiedergeben.

Eine effiziente Implementierung dieses Schemas, welches höchstens eine Multiplikation je Probe erfordert, ist in Gleichungen 33 und 34 angegeben.

Die erste Aufteilungsperiode der Länge N_d ist durch Gleichung 33 angegeben:

$$y_n = x_n \qquad 0 \leq n < N_1 - W$$

$$y_n = x_n + [x_{n+\Delta} - x_n] * (n - N_1 + W + 1)/(W+1) \qquad N_1 - W \leq n < N_d$$

Gleichung 33

Die zweite Pitchperiode der Länge N_r wird erzeugt, wie es in Gleichung 34 gezeigt ist:

$$y_n = x_{n-\Delta} + [x_n - x_{n-\Delta}] * (n - \Delta - N_1 + W + 1)/(W+1) \qquad N_1 \leq n < N_1 + \Delta$$

$$y_n = x_n \qquad N_{1+\Delta} \leq n < N_1 + N_r$$

Gleichung 34

Wie aus Fig. 14 ersichtlich, ist die Sequenz L_n im wesentlichen bis zu dem Punkt N_1-W gleich der ersten Pitchperiode. Bei diesem Punkt wird eine abnehmende Rampe WL auf das Signal angewendet, um den Einfluß der ersten Pitchperiode zu dämpfen.

Wie ferner erkannt werden kann, beginnt die Gewichtungsfunktion WR an dem Punkt $N_1 - W + \Delta$ und wendet eine ansteigende Rampe auf die Sequenz x_n bis zu dem Punkt $N_1 + \Delta$ an. Von diesem Punkt wird ein konstanter Wert angewendet. Dies bewirkt eine Dämpfung des Einflusses der rechten Sequenz und betont die linke während des Anfangs der Gewichtungsfunktionen und generiert ein Endungssegment, welches im wesentlichen gleich ist dem Endungssegment von x_n, wobei die rechte Sequenz betont wird, während die linke gedämpft wird. Wenn die zwei Funktionen vermischt sind, ist die resultierende Wellenform y_n im wesentlichen gleich dem Anfang

von x_n an dem Anfang der Sequenz, wobei an dem Punkt $N_1 - W$ eine modifizierte Sequenz bis zu dem Punkt N_1 generiert wird. Von N_1 zu der Endung resultiert die Sequenz x_n als um Δ verschoben.

Es besteht also ein Bedarf für das Einfügen von Pitchperioden, um die Dauer eines gegebenen Tones bzw. Lautes zu verlängern. Eine Pitchperiode wird gemäß dem in Fig. 15 gezeigten Algorithmus eingeführt, wobei Bezug genommen wird auf die Zeichnungen von Fig. 16.

Der Algorithmus beginnt, indem er ein Steuersignal empfängt, und fügt eine Pitchperiode zwischen Rahmen L_n und R_n ein (Block 450). Anschließend werden sowohl L_n als auch R_n in dem Puffer gespeichert (Block 451), wobei L_n und R_n zwei benachbarte Pitchperioden eines Sprachdiphons sind. (Ohne Verlust der Generalisierung wird für die Beschreibung angenommen, daß die zwei Sequenzen gleiche Längen N aufweisen.)

Um eine Pitchperiode x_n derselben Dauer einzufügen, ohne eine Diskontinuität zwischen L_n und x_n und zwischen x_n und R_n zu veranlassen, sollte die Pitchperiode x_n in der Nähe von n=0 R_n ähneln (Beibehaltung der Kontinuität von L_n zu x_n), und sollte in der Nähe von n=N L_n ähneln (Beibehaltung der Kontinuität von x_n zu R_n). Dies wird erreicht, indem x_n definiert wird, wie es in Gleichung 35 gezeigt ist:

$$x_n = R_n + (L_n - R_n) * [(n + 1)/(N + 1)] \qquad 0 \leq n < N-1$$

Gleichung 35

Konzeptionsmäßig, wie in Fig. 15 gezeigt, schreitet der Algorithmus fort, indem ein linker Vektor WL (L_n) erzeugt

wird, im wesentlichen anwendend die ansteigende Rampe bzw. auf die ansteigende Rampe WL auf das Signal L_n (Block 452).

Ein rechter Vektor WR (R_n) wird unter Verwendung des Gewichtungsvektors WR (Block 453) generiert, welcher im wesentlichen eine abnehmende Rampe ist, wie in Fig. 16 gezeigt. Demzufolge wird die Endung von L_n mit dem linken Vektor betont, wobei der Anfang von R_n mit dem Vektor WR betont wird.

Anschließend werden WR (L_n) und WR (R_n) vermischt um eine eingefügte Periode x_n zu erzeugen (Block 454).

Die Berechnungsanforderung zum Einfügen einer Pitchperiode entspricht somit lediglich einer Multiplikation und zwei Additionen je Sprachprobe.

Schließlich erzeugt die Verkettung von L_n, x_n und R_n eine Sequenz mit einer eingefügten Pitchperiode (Block 455).

Das Löschen einer Pitchperiode wird wie in Fig. 17 unter Bezugnahme auf die Graphen von Fig. 18 gezeigt, ausgeführt. Dieser Algorithmus, welcher sehr ähnlich zu dem Algorithmus zum Einfügen einer Pitchperiode ist, beginnt mit dem Empfangen eines Steuersignales, welches die Löschung einer Pitchperiode R_n angibt, und zwar der L_n folgenden (Block 500). Anschließend werden die Pitchperioden L_n und R_n in dem Puffer gespeichert (Block 501). Dies ist bildartig in Fig. 18 an dem oberen Rand der Seite dargestellt. Erneut ohne Einfluß auf die Generalisierung wird angenommen, daß die zwei Sequenzen gleiche Längen N aufweisen.

Der Algorithmus wird abgearbeitet, um die Pitchperiode L_n zu verändern, welche (der zu löschenden) R_n vorangeht, so daß sie R_n ähnelt, wenn n sich an N annähert. Dies erfolgt, wie angegeben in Gleichung 36:

$$L_n = L_n + (R_n - L_n) * [(n+1)/(N+1)] \qquad 0 \leq n < N-1$$

Gleichung 36

In Gleichung 36 ist die resultierende Sequenz L_n am unteren Rand von Fig. 18 gezeigt. Konzeptionsmäßig wendet die Gleichung 36 eine Gewichtungsfunktion WL auf die Sequenz L_n an (Block 502). Dies betont wie dargestellt den Anfang der Sequenz L_n. Anschließend wird durch Anwendung eines Gewichtungsvektors WR auf die Sequenz R_n, ein rechter Vektor WR (R_n) erzeugt, welcher die Endung von R_n betont (Block 503).

WL (L_n) und WR (R_n) werden vermischt um den resultierenden Vektor L_n zu erzeugen (Block 504). Schließlich wird die Sequenz L_n-R_n durch die Sequenz L_n in dem Pitchperiodenstring ersetzt (Block 505).

V. Folgerungen

Dementsprechend schlägt die vorliegende Erfindung ein Nur-Software-Text-zu-Sprach-System vor, welches effizient ist, sehr geringe Mengen an Speicher verwendet und auf eine große Vielzahl an Standard-Mikrocomputerplattformen portierbar ist. Sie nutzt die Kenntnis bezüglich Sprachdaten, wobei eine Sprachkompressions-, Vermischungs- und Dauersteuerroutine erzeugt wird, um sehr hohe Sprachqualität mit sehr geringen Berechnungsressourcen zur Verfügung zu stellen.

Ein Source-Code-Listing der Software zum Ausführen der Kompression und Dekompression, des Vermischens, sowie die Dauer und Pitchsteuerroutinen sind als Anhang und als Beispiel einer bevorzugten Ausführungsform der vorliegenden Erfindung bereitgestellt.

Die vorangegange Beschreibung von bevorzugten Ausführungsformen der Erfindung wurde angegeben zum Zwecke der illustrativen Darstellung und Beschreibung. Sie ist nicht als abschließend zu erachten oder als die Erfindung auf die präzisen offenbarten Formen beschränkend. Offensichtlich können viele Veränderungen und Modifikationen von dem Durchschnittsfachmann durchgeführt werden. Die Ausführungsformen wurden gewählt und beschrieben, um am besten bzw. einfachsten die Prinzipien der Erfindung zu erläutern, sowie deren praktikable Anwendung, so daß der Fachmann in die Lage versetzt wird, die Erfindung für verschiedene Ausführungsformen zu verstehen, wobei verschiedene Modifikationen geeignet sein können für die spezifische beabsichtigte Verwendung bzw. Anwendung. Der Umfang der Erfindung soll durch die folgenden Ansprüche definiert sein.

APPENDIX

37 C.F.R. §1.96(a)

COMPUTER PROGRAM LISTINGS

TABLE OF CONTENTS

I. ENCODER MODULE

```
#include <stdio.h>
#include <math.h>
#include <StdLib.h>
#include <types.h>
#include <fcntl.h>
#include <string.h>

#include <types.h>
#include <files.h>
#include <resources.h>
#include <memory.h>
#include "vqcoder.h"

#define    LAST_FRAME_FLAG          128
#define    PBUF_SIZE  440
static  float          oc_state[2], nsf_state[NSF_ORDER+1];
static  short          pstate[PORDER+1], dstate[PORDER+1];
static  short          AnaPbuf[PBUF_SIZE];

static  short          vsize, cbook_size, bs_size;

#pragma segment vqlib

/* Read Code Books */
float   *EncodeBook[MAX_CBOOK_SIZE];
short   *DecodeBook[MAX_CBOOK_SIZE];
get_cbook(short ratio)
{
   short *p;
   short    frame_size, i;
   static   short last_ratio = 0;

   Handle h;
   int      skip;
   h = GetResource('CBOK',1);
   HLock(h);
   p = (short *) *h;

   if (ratio = = last_ratio)
        return;
   last_ratio = ratio;

   if (ratio < 3)
        return;

   if (NOMINAL_PITCH < 165)
```

```
        frame_size = 96;
    else
        frame_size = 160;

    get_compr_pars(ratio, frame_size, &vsize, &cbook_size, &bs_size);
    skip = 0;
    while (p[skip+1] != vsize)
    {
        short t1, t2;
        t2 = p[skip];
        t1 = p[skip+1];
        skip += sizeof(float) * (2 * t2-1) * (t1 + 1) / sizeof(short)
            + (2 * t2 * t1 + 2);
    }

    /*Skip Binary search tree */
    skip += sizeof(float) * (cbook_size-1) * (vsize+1) / sizeof(short)
        + (cbook_size * vsize + 2);

    /* Get pointers to Full search code books */
    for (i = 0; i < cbook_size; i++)
    {
        EncodeBook[i] = (float *) &p[skip];
        skip += (vsize+1) * sizeof (float) / sizeof(short);
    }

    for (i = 0; i < cbook_size; i++)
    {
        DecodeBook[i] = p + skip;
        skip += vsize;
    }
}

char *getcbook(long *len, short ratio)
{
    get_cbook(ratio);
    *len = sizeof(short) * vsize * cbook_size;
        /* plus one is to make space at the end for the array of pointers */
    return (char*) DecodeBook[0];
}

/* A Routine for Pitch filter parameter Estimation */
GetPitchFilterPars (x, len, pbuf, min_pitch, max_pitch, pitch, beta)
float   *beta;
short   *x, *pbuf;
short   min_pitch, max_pitch;
short   len;
unsigned int *pitch;
{
    /* Estimate long-term predictor */
```

```
    int     best_pitch, i, j;
    float   syy, sxy, best_sxy = 0.0, best_syy = 1.0;
    short   *ptr;

    best_pitch = min_pitch;
    ptr = pbuf + PBUF_SIZE - min_pitch;
    syy = 1.0;
    for (i = 0; i < len; i++)
    {
            syy += (*ptr) * (*ptr);
            ptr++;
    }
    for (j = min_pitch; j < max_pitch; j++)
    {
        sxy = 0.0;
        ptr = pbuf + PBUF_SIZE - j;
        for (i = 0; i < len; i++)
            sxy += x[i] * (*ptr++);

        if (sxy > 0 && (sxy * sxy * best_syy > best_sxy * best_sxy * syy))
        {
            best_syy = syy;
            best_sxy = sxy;
            best_pitch = j;
        }
        syy = syy - pbuf[PBUF_SIZE - j + len - 1] * pbuf[PBUF_SIZE - j + len - 1]
                + pbuf[PBUF_SIZE - j - 1] * pbuf[PBUF_SIZE - j - 1];
    }

    *pitch = best_pitch;
    *beta = best_sxy / best_syy;
}

/* Quantization of LTP gain parameter */
CodePitchFilterGain(beta, bcode)
float beta;
unsigned int *bcode;
{
    int i;
    for (i = 0; i < DLB_TAB_SIZE; i++)
    {
        if (beta <= dlb_tab[i])
            break;
    }
    *bcode = i;
}

/* Pitch filter */
PitchFilter(data, len, pbuf, pitch, ibeta)
float   *data;
```

```
short  ibeta;
short  *pbuf;
short      len;
unsigned int pitch;
{

  long          pn;
  int       i, j;


   j = PBUF_SIZE - pitch;
   for (i = 0; i < len; i++)
   {
       pn  = ((ibeta * pbuf[j++]) >> 4);
       data[i] -= pn;
   }
}

/* Forward Noise Shaping filter */
FNSFilter(float *inp, float *state, short len, float *out)
{
   short i, j;
   for (j = 0; j < len; j++)
   {
       float tmp = inp[j];
       for (i = 1; i <= NSF_ORDER; i++)
           tmp += state[i] * nsf[i];
       out[j] = state[0] = tmp;
       for (i = NSF_ORDER; i > 0; i--)
           state[i] = state[i-1];
   }
}

/* Update Noise shaping Filter states */
UpdateNSFState(float *inp, float *state, short len)
{
   short i, j;
   float     temp_state[NSF_ORDER+1];

   for (i = 0; i <= NSF_ORDER; i++)
       temp_state[i] = 0;

   for (j = 0; j < len; j++)
   {
       float tmp = inp[j];
       for (i = 1; i <= NSF_ORDER; i++)
           tmp += temp_state[i] * nsf[i];
       temp_state[0] = tmp;
       for (i = NSF_ORDER; i > 0; i--)
           temp_state[i] = temp_state[i-1];
```

```
    }
    for (i = 0; i < = NSF_ORDER; i++)
        state[i] = state[i] - temp_state[i];
}

/* Quantization of Segment Power */
CodeBlockGain(power, gcode)
float power;
unsigned int *gcode;
{
    int i;
    for (i = 0; i < DLG_TAB_SIZE; i++)
    {
        if (power < = dlg_tab[i])
            break;
    }
    *gcode = i;
}

/* Full search Coder */
VQCoder(float *x, float *nsf_state, short len, struct frame *bs)
{
    float           max_x, tmp;
    int         i, j, k, index, lshift_count;
    unsigned int    gcode;
    float           min_err = 0;

    max_x = x[0];
    for (i = 1; i < len; i++)
        if ( fabs(x[i]) > max_x)
            max_x = fabs(x[i]);

    CodeBlockGain(max_x, &gcode);
    max_x = qlg_tab[gcode];
    lshift_count = 7 - gcode;       /* To scale 14-bit Code book output to the 16-bit
actual value */
    bs->gcode = gcode;

    for (i = 0; i < len; i += vsize)
    {
        /* Filter the data vector */
        FNSFilter(&x[i], nsf_state, vsize, &x[i]);

        /* Scale data */
        for (j = i; j < i + vsize; j++)
            x[j] = x[j] * 1024 / max_x;

        index = 0;
        for (j = 0; j < cbook_size; j++)
        {
```

```
            tmp = EncodeBook[j][vsize] * 1024.0;
            for (k = 0; k < vsize; k++)
                tmp -= x[i+k] * EncodeBook[j][k];

            if (tmp < min_err || j == 0)
            {
                index = j;
                min_err = tmp;
            }
        }
        bs->vqcode[i/vsize] = index;

        /* Rescale data: Decoded data is 14-bits, convert to 16 bits */
        if (lshift_count)
        {
            for (k = 0; k < vsize; k++)
                x[i+k] = ((4 * DecodeBook[index][k]) >> lshift_count);
        }
        else
        {
            for (k = 0; k < vsize; k++)
                x[i+k] = 4 * DecodeBook[index][k];
        }

        /* Update noise shaping filter state */
        UpdateNSFState(&x[i], nsf_state, vsize);
    }
}

init_compress()
{
    int i;
    oc_state[0] = 0;;
    oc_state[1] = 0;;
    for (i = 0; i <= PORDER; i++)
        pstate[i] = dstate[i] = 0;
    for (i = 0; i < PBUF_SIZE; i++)
        AnaPbuf[i] = 0;
    for (i=0; i <= NSF_ORDER; i++)
        nsf_state[i] = 0;
}

Encoder(xn, frame_size, min_pitch, max_pitch, bs)
short xn[];
struct frame *bs;
short   frame_size, min_pitch, max_pitch;
{
    unsigned int pitch, bcode;
    float    preemp_xn[PBUF_SIZE], beta;
    short    xn_copy[PBUF_SIZE];
```

```
short    ibeta;
float    acc;
int i, j;


/* Offset Compensation */
for (i = 0; i < frame_size; i++)
{
    float inp = xn[i];
    xn[i] = inp - oc_state[0] + ALPHA * oc_state[1];
    oc_state[1] = xn[i];
    oc_state[0] = inp;
}

/* Linear Prediction Filtering */
for (i = 0; i < frame_size; i++)
{
    acc = pstate[0] = xn[i];
    for (j = 1; j <= PORDER; j++)
        acc -= pstate[j] * pfilt[j];
    xn_copy[i] = preemp_xn[i] = acc;
    for (j = PORDER; j > 0; j--)
        pstate[j] = pstate[j-1];
}

GetPitchFilterPars (xn_copy, frame_size, AnaPbuf, min_pitch,
    max_pitch, &pitch, &beta);
CodePitchFilterGain(beta, &bcode);
ibeta = qlb_tab[bcode];

bs->bcode = bcode;
bs->pitch = pitch - min_pitch + 1;

PitchFilter(preemp_xn, frame_size, AnaPbuf, pitch, ibeta);

VQCoder(preemp_xn, nsf_state, frame_size, bs);

/* Inverse Filtering */
j = PBUF_SIZE - pitch;
for (i = 0; i < frame_size; i++)
{
    xn_copy[i] = preemp_xn[i];
    xn_copy[i] += ((ibeta * AnaPbuf[j++]) >> 4);
}

/* Update Pitch Buffer */
j = 0;
for (i = frame_size; i < PBUF_SIZE; i++)
    AnaPbuf[j++] = AnaPbuf[i];
for (i = 0; i < frame_size; i++)
```

```
        AnaPbuf[j++] = xn_copy[i];

    /* Inverse LP filtering  */
    for (i = 0; i < frame_size; i++)
    {
        acc = xn_copy[i];
        for (j = 1; j <= PORDER; j++)
            acc = acc + dstate[j] * pfilt[j];
        dstate[0] = acc;
        for (j = PORDER; j > 0; j--)
            dstate[j] = dstate[j-1];
    }

    for (j = 0; j <= PORDER; j++)
        pstate[j] = dstate[j];
}

compress (short *input, short ilen, unsigned char *output, long *olen, long docomp)
{
    int             i, j, vcount;
    unsigned char   temp;
    short           frame_size, min_pitch, max_pitch;

    if (docomp > 2)
    {
        init_compress();

        if (NOMINAL_PITCH < 165)
        {
            min_pitch = 96;
            frame_size = 96;
            max_pitch = 350;
        }
        else
        {
            min_pitch = 160;
            frame_size = 160;
            max_pitch = 414;
        }

        bs_size = frame_size / vsize + 2;
        /* TEMPORARY: Storing State information */
        pstate[1] = *(input - 1);
        if (pstate[1] > 0)
            pstate[1] = (pstate[1] + 128) / 256 + 128;
        else
            pstate[1] = (pstate[1] - 128) / 256 + 128;

        if (pstate[1] < 0)
            pstate[1] = 0;
```

```
        if (pstate[1] > 255)
            pstate[1] = 255;
        *output = pstate[1];
        j = 1;
        pstate[1] = pstate[1] - 128;
        pstate[1] = 256 * pstate[1];
        dstate[1] = pstate[1];
        /* End of Hack */
        for (i = 0; i < ilen; i += frame_size)
        {
            Encoder(input+i, frame_size, min_pitch, max_pitch, output+j);
            j += bs_size;
        }
        j -= bs_size;

        /* Number of vectors in last frame */
        vcount = (ilen + frame_size - i + vsize - 1) / vsize;
        temp = output[j];
        output[j] = vcount + LAST_FRAME_FLAG;
        output[j + vcount + 2] = temp;
        *olen = j + vcount + 3;
    }
    else
    {
        static long SampCount = 0;
        copy(input, output, 2*ilen);
        SampCount += ilen;
        *olen = ilen;
    }
}

copy(a, b, len)
short  *a, *b;
short  len;
{
    int i;
    for (i = 0; i < len; i++)
        *b++ = (*a++);
}
```

II. DECODER MODULE

```
#include <Types.h>
#include <Memory.h>
#include <Quickdraw.h>
#include <ToolUtils.h>
#include <errors.h>
#include <files.h>

#include "vtcint.h"
#include <stdlib.h>
#include <math.h>
#include <sysequ.h>
#include <string.h>

#define MAX_CBOOK_SIZE          256
#define   LAST_FRAME_FLAG       128
#define   PORDER                  1
#define   IPCONS                  7                     /* 7/8 */

#define   LARGE_NUM               100000000
#define   VOICED    1

#define LEFT                  0
#define   RIGHT                   1
#define   UNVOICED                0

#define   PFILT_ORDER             8

struct frame {
  unsigned  gcode : 4;
  unsigned      bcode : 4;
  unsigned pitch : 8;
  unsigned char vqcode[];
};

void expand(short **DecodeBook, short frame_size, short vsize,
   short min_pitch, struct frame *bs, short *output, short smpnum);

get_compr_pars(short ratio, short frame_size, short *vsize,
   short *cbook_size, short *bs_size)
{
   switch (ratio)
   {
       case 4:
           *vsize = 2;
           *cbook_size = 256;
           *bs_size = frame_size/2 + 2;
           break;
```

```
        case 7:
            *vsize = 4;
            *cbook_size = 256;
            *bs_size = frame_size/4 + 2;
            break;
        case 14:
            *vsize = 8;
            *cbook_size = 256;
            *bs_size = frame_size/8 + 2;
            break;
        case 24:
            *vsize = 16;
            *cbook_size = 256;
            *bs_size = frame_size/16 + 2;
            break;
        default:
            *vsize = 2;
            *cbook_size = 256;
            *bs_size = frame_size/2 + 2;
            break;
    }
}

short *SnInit(short comp_ratio)
{
    short *state, *ptr;
    int i;

    state = ptr = (short*)NewPtr((PFILT_ORDER+1 + PFILT_ORDER/2 + 2) *
sizeof(short));
    if ( state == nil )
    {
        return nil;
    }
    for (i=0;i<PFILT_ORDER+1;i++)
        *ptr++ = 0;
/*
    if (comp_ratio == 24)
    {
        *ptr++ = 0.036953 * 32768 + 0.5;
        *ptr++ = -0.132232 * 32768 - 0.5;
        *ptr++ = 0.047798 * 32768 + 0.5;
        *ptr++ = 0.403220 * 32768 + 0.5;
        *ptr++ = 0.290033 * 32768 + 0.5;
    }
    else
    {
        *ptr++ = 0.074539 * 32768 + 0.5;
        *ptr++ = -0.174290 * 32768 - 0.5;
        *ptr++ = 0.013704 * 32768 + 0.5;
```

```
        *ptr++ = 0.426815 * 32768 + 0.5;
        *ptr++ = 0.320707 * 32768 + 0.5;
    }
*/
    if (comp_ratio == 24)
    {
        *ptr++ = 1211;
        *ptr++ = -4333;
        *ptr++ = 1566;
        *ptr++ = 13213;
        *ptr++ = 9504;
    }
    else
    {
        *ptr++ = 2442;
        *ptr++ = -5711;
        *ptr++ = 449;
        *ptr++ = 13986;
        *ptr++ = 10509;
    }
    *ptr = 0;        /* DC value */
    return state;
}

SnDone(char *state)
{
    if ( state != nil )
    {
        DisposPtr(state);
    }
}

short **SnDeInit(p, ratio,  frame_size)
short *p,ratio, frame_size;
{
    int i;
    short cbook_size = 256, vsize = 16, bs_size;
    short **DecodeBook;

    get_compr_pars(ratio, frame_size, &vsize, &cbook_size, &bs_size);

    DecodeBook = (short**)NewPtr(cbook_size * sizeof(short*));
    if (DecodeBook) {
        for (i = 0; i < cbook_size; i++)
        {
            DecodeBook[i] = p;
            p += vsize;
        }
    }
    return DecodeBook;
```

```
}

SnDeDone(char *DecodeBook)
{
   if ( DecodeBook != nil )
   {
       DisposPtr(DecodeBook);
   }
}

void
expand(short **DecodeBook, short frame_size, short vsize,
   short min_pitch, struct frame *bs, short *output, short smpnum)
{
   short    count;
   short    *bptr, *sptr1, *sptr2;
   unsigned short pitch, bcode;
/*
   short qlb_tab[] = {
   1, 2, 3, 4, 5, 6, 7, 8,
   9, 10, 11, 12, 13, 14, 15, 16
   };
*/
   bcode = bs->bcode;
   pitch = bs->pitch + min_pitch - 1;

   /* Decode VQ vectors */
   {
       unsigned    char     *cptr;
       short    k, vsize_by_2;
       short    rshift_count = 7 - bs->gcode;    /* We want the output to be 14-bit
number */

       sptr1 = output + smpnum;
       cptr = bs->vqcode;
       vsize_by_2 = (vsize >> 1) + 1; /* +1 since we do a while (--i) instead of
while (i--) */
       if (rshift_count)
       {
           for (k = 0; k < frame_size; k += vsize)
           {
               bptr = DecodeBook[*cptr++];
               count = vsize_by_2;
               while (--count)
               {
                   *sptr1++ = ((*bptr++) >> rshift_count);
                   *sptr1++ = ((*bptr++) >> rshift_count);
               }
           }
       }
```

```
        else
        {
            for (k = 0; k < frame_size; k += vsize)
            {
                bptr = DecodeBook[*cptr++];
                count = vsize_by_2;
                while (--count)
                {
                    *sptr1++ = *bptr++;
                    *sptr1++ = *bptr++;
                }
            }
        }
    }

    /* Inverse Filtering */
    if (smpnum < pitch)
    {
        sptr1 = output + pitch;
        count = smpnum + frame_size + 1 - pitch; /* +1 since we do a while (--i)
instead of while (i--) */
        sptr2 = sptr1 - pitch;
        switch (bcode)
        {
            case 0:
                while (--count)
                    *sptr1++ += ((*sptr2++) >> 4);
                break;
            case 1:
                while (--count)
                    *sptr1++ += ((*sptr2++) >> 3);
                break;
            case 2:
                while (--count)
                    *sptr1++ += ((3 * (*sptr2++)) >> 4);
                break;
            case 3:
                while (--count)
                    *sptr1++ += ((*sptr2++) >> 2);
                break;
            case 4:
                while (--count)
                    *sptr1++ += ((5 * (*sptr2++)) >> 4);
                break;
            case 5:
                while (--count)
                    *sptr1++ += ((3 * (*sptr2++)) >> 3);
                break;
            case 6:
                while (--count)
```

```
            *sptr1++ += ((7 * (*sptr2++)) >> 4);
        break;
    case 7:
        while (--count)
            *sptr1++ += ((*sptr2++) >> 1);
        break;
    case 8:
        while (--count)
        {
            long    tmp;
            tmp = *sptr2++;
            *sptr1++ += (((tmp << 3) + tmp) >> 4);
        }
        break;
    case 9:
        while (--count)
            *sptr1++ += ((5 * (*sptr2++)) >> 3);
        break;
    case 10:
        while (--count)
        {
            long    tmp;
            tmp = *sptr2++;
            *sptr1++ += (((tmp << 3) + 3 * tmp) >> 4);
        }
        break;
    case 11:
        while (--count)
            *sptr1++ += ((3 * (*sptr2++)) >> 2);
        break;
    case 12:
        while (--count)
        {
            long    tmp;
            tmp = *sptr2++;
            *sptr1++ += (((tmp << 4) - 3 * tmp) >> 4);
        }
        break;
    case 13:
        while (--count)
            *sptr1++ += ((7 * (*sptr2++)) >> 3);
        break;
    case 14:
        while (--count)
        {
            long    tmp;
            tmp = *sptr2++;
            *sptr1++ += (((tmp << 4) - tmp) >> 4);
        }
        break;
```

```
        case 15:
            while (--count)
                *sptr1++ += *sptr2++;
            break;
    }
} else {
    sptr1 = output + smpnum;
    sptr2 = sptr1 - pitch;
    count = (frame_size / 4) + 1;
    switch (bcode)
    {
        case 0:
            while (--count) {
                *sptr1++ += ((*sptr2++) >> 4);
                *sptr1++ += ((*sptr2++) >> 4);
                *sptr1++ += ((*sptr2++) >> 4);
                *sptr1++ += ((*sptr2++) >> 4);
            }
            break;
        case 1:
            while (--count) {
                *sptr1++ += ((*sptr2++) >> 3);
                *sptr1++ += ((*sptr2++) >> 3);
                *sptr1++ += ((*sptr2++) >> 3);
                *sptr1++ += ((*sptr2++) >> 3);
            }
            break;
        case 2:
            while (--count) {
                *sptr1++ += ((3 * (*sptr2++)) >> 4);
                *sptr1++ += ((3 * (*sptr2++)) >> 4);
                *sptr1++ += ((3 * (*sptr2++)) >> 4);
                *sptr1++ += ((3 * (*sptr2++)) >> 4);
            }
            break;
        case 3:
            while (--count) {
                *sptr1++ += ((*sptr2++) >> 2);
                *sptr1++ += ((*sptr2++) >> 2);
                *sptr1++ += ((*sptr2++) >> 2);
                *sptr1++ += ((*sptr2++) >> 2);
            }
            break;
        case 4:
            while (--count) {
                *sptr1++ += ((5 * (*sptr2++)) >> 4);
                *sptr1++ += ((5 * (*sptr2++)) >> 4);
                *sptr1++ += ((5 * (*sptr2++)) >> 4);
                *sptr1++ += ((5 * (*sptr2++)) >> 4);
            }
```

```
    break;
case 5:
    while (--count) {
        *sptr1++ += ((3 * (*sptr2++)) >> 3);
        *sptr1++ += ((3 * (*sptr2++)) >> 3);
        *sptr1++ += ((3 * (*sptr2++)) >> 3);
        *sptr1++ += ((3 * (*sptr2++)) >> 3);
    }
    break;
case 6:
    while (--count) {
        *sptr1++ += ((7 * (*sptr2++)) >> 4);
        *sptr1++ += ((7 * (*sptr2++)) >> 4);
        *sptr1++ += ((7 * (*sptr2++)) >> 4);
        *sptr1++ += ((7 * (*sptr2++)) >> 4);
    }
    break;
case 7:
    while (--count) {
        *sptr1++ += ((*sptr2++) >> 1);
        *sptr1++ += ((*sptr2++) >> 1);
        *sptr1++ += ((*sptr2++) >> 1);
        *sptr1++ += ((*sptr2++) >> 1);
    }
    break;
case 8:
    while (--count) {
        long    tmp;
        tmp = *sptr2++;
        *sptr1++ += ((8 * tmp + tmp) >> 4);
        tmp = *sptr2++;
        *sptr1++ += ((8 * tmp + tmp) >> 4);
        tmp = *sptr2++;
        *sptr1++ += ((8 * tmp + tmp) >> 4);
        tmp = *sptr2++;
        *sptr1++ += ((8 * tmp + tmp) >> 4);
    }
    break;
case 9:
    while (--count) {
        *sptr1++ += ((5 * (*sptr2++)) >> 3);
        *sptr1++ += ((5 * (*sptr2++)) >> 3);
        *sptr1++ += ((5 * (*sptr2++)) >> 3);
        *sptr1++ += ((5 * (*sptr2++)) >> 3);
    }
    break;
case 10:
    while (--count) {
        long tmp;
        tmp = *sptr2++;
```

```
            *sptr1++ += (((tmp << 3) + 3 * tmp) >> 4);
            tmp = *sptr2++;
            *sptr1++ += (((tmp << 3) + 3 * tmp) >> 4);
            tmp = *sptr2++;
            *sptr1++ += (((tmp << 3) + 3 * tmp) >> 4);
            tmp = *sptr2++;
            *sptr1++ += (((tmp << 3) + 3 * tmp) >> 4);
        }
        break;
    case 11:
        while (--count) {
            *sptr1++ += ((3 * (*sptr2++)) >> 2);
            *sptr1++ += ((3 * (*sptr2++)) >> 2);
            *sptr1++ += ((3 * (*sptr2++)) >> 2);
            *sptr1++ += ((3 * (*sptr2++)) >> 2);
        }
        break;
    case 12:
        while (--count) {
            long tmp;
            tmp = *sptr2++;
            *sptr1++ += (((tmp << 4) - 3 * tmp) >> 4);
            tmp = *sptr2++;
            *sptr1++ += (((tmp << 4) - 3 * tmp) >> 4);
            tmp = *sptr2++;
            *sptr1++ += (((tmp << 4) - 3 * tmp) >> 4);
            tmp = *sptr2++;
            *sptr1++ += (((tmp << 4) - 3 * tmp) >> 4);
        }
        break;
    case 13:
        while (--count) {
            *sptr1++ += ((7 * (*sptr2++)) >> 3);
            *sptr1++ += ((7 * (*sptr2++)) >> 3);
            *sptr1++ += ((7 * (*sptr2++)) >> 3);
            *sptr1++ += ((7 * (*sptr2++)) >> 3);
        }
        break;
    case 14:
        while (--count) {
            long tmp;
            tmp = *sptr2++;
            *sptr1++ += (((tmp << 4) - tmp) >> 4);
            tmp = *sptr2++;
            *sptr1++ += (((tmp << 4) - tmp) >> 4);
            tmp = *sptr2++;
            *sptr1++ += (((tmp << 4) - tmp) >> 4);
            tmp = *sptr2++;
            *sptr1++ += (((tmp << 4) - tmp) >> 4);
        }
```

```
            break;
        case 15:
            while (--count) {
                *sptr1++ += *sptr2++;
                *sptr1++ += *sptr2++;
                *sptr1++ += *sptr2++;
                *sptr1++ += *sptr2++;
            }
            break;
    }
  }

}

short SnDecompress(DecodeBook, ratio, frame_size, min_pitch, bstream, output)
short **DecodeBook, ratio;
unsigned char *bstream;
short *output, frame_size, min_pitch;
{
    short    count, SampCount;
    register short    dstate;
    short    vcount;
    short    vsize, cbook_size, bs_size;

    get_compr_pars(ratio, frame_size, &vsize, &cbook_size, &bs_size);

    dstate = *bstream++;
    dstate = (dstate - 128) << 6;

    SampCount = 0;
    while((*bstream & LAST_FRAME_FLAG) == 0)
    {
        expand(DecodeBook, frame_size, vsize, min_pitch,
            (struct frame *)bstream, output, SampCount);
        bstream += bs_size;
        SampCount += frame_size;
    }
    vcount = *bstream - LAST_FRAME_FLAG;
    *bstream = *(bstream + 2 + vcount);
    expand(DecodeBook, frame_size, vsize, min_pitch,
        (struct frame *)bstream, output, SampCount);
    *bstream = vcount + LAST_FRAME_FLAG;
    SampCount += vcount * vsize;

    count = (SampCount >> 1) + 1;
    while (--count) {
        *output++ = dstate = ((IPCONS * dstate) >> 3) + *output;
        *output++ = dstate = ((IPCONS * dstate) >> 3) + *output;
    }
    output -= SampCount;
```

```
    return SampCount;
}

#define    FILTER  state + PFILT_ORDER+1
#define DC_VAL    state + PFILT_ORDER + PFILT_ORDER/2 + 2
void SnSampExpandFilt(short *src, short off, short len,
    char *dest,short *state)
{
    short          input, temp;
    long           acc;
    register short dc = *(DC_VAL);
    register short *sptr1, *sptr2;

    src += off;
    len++;
    sptr1 = state;
    sptr2 = state + PFILT_ORDER;
    while (--len) {
        input = *src++ - dc;
        dc += input >> 5;

        temp = input + *sptr1++; /* (state[0] + state[8]) * filter[0] */
        acc = temp * *(FILTER);

        temp = *--sptr2 + *sptr1++;  /* (state[1] + state[7]) * filter[1] */
        acc += temp * *(FILTER+1);

        temp = *--sptr2 + *sptr1++;  /* (state[2] + state[6]) * filter[2] */
        acc += temp * *(FILTER+2);

        temp = *--sptr2 + *sptr1++;  /* (state[3] + state[5]) * filter[3] */
        acc += temp * *(FILTER+3);

        acc += *sptr1 * *(FILTER+4); /* state[4] * filter[4] */

        if (acc > 0)
        {
            temp = (acc + (257 << 20)) >> 21;
            if (temp > 255)
                temp = 255;
        }
        else
        {
            temp = (acc + (255 << 20)) >> 21;
            if (temp < 0)
                temp = 0;
        }
        *dest++ = temp;
```

```
        sptr1 -= 4;
        sptr2 -= 4;
        *sptr1++ = *sptr2++;    /* state[0] = state[1] */
        *sptr1++ = *sptr2++;    /* state[1] = state[2] */
        *sptr1++ = *sptr2++;    /* state[2] = state[3] */
        *sptr1++ = *sptr2++;    /* state[3] = state[4] */
        *sptr1++ = *sptr2++;    /* state[4] = state[5] */
        *sptr1++ = *sptr2++;    /* state[5] = state[6] */
        *sptr1++ = *sptr2++;    /* state[6] = state[7] */
        *sptr1 = input;         /* state[7] = input */
        sptr1 -= 7;
    }
    *(DC_VAL) = dc;
}
```

III. BLENDING MODULE

```
/* A module for blending two diphones */

typedef struct {
   short lptr, pitch;
   short weight, weight_inc;
} bstate;

void SnBlend(pitchp lp, pitchp rp, short cur_tot, short tot,
   short type, bstate *bs)
{
#pragma unused (tot)

   short    count;
   short    *ptr1, *ptr2;

   if (type == VOICED)
   {
      if (cur_tot)
         return;
      {
         short    weight;
         long     min_amdf;
         short    best_lag = 0, lag;
         short    window_size;
         short    weight_inc;

         /* First replicate the left pitch period */
         ptr1 = lp->bufp;
         ptr2 = ptr1 + lp->olen;
         count = lp->olen + 1;
         while (--count)
            *ptr2++ = *ptr1++;

         /* Smooth the discontinuity */
         {
            register short en, e2;

            en = lp->bufp[2] +
               3 * (lp->bufp[0] - lp->bufp[1]) - lp->bufp[lp->olen - 1];

            e2 = lp->bufp[0] - lp->bufp[lp->olen - 1];


            if (en * en > e2 * e2)
               en = e2;
```

```
    ptr2 = lp->bufp + lp->olen;
    count = (lp->olen >> 1) + 1;
    while (--count)
    {
        *--ptr2 += en;
        en = (((en << 4) - en) >> 4);
    }
}

min_amdf = LARGE_NUM;

window_size = rp->olen;
if (lp->olen < rp->olen)
    window_size = lp->olen;

lag = rp->olen;
while (--lag)
{
    long amdf = 0;
    ptr1 = rp->bufp;
    ptr2 = lp->bufp + lag;
    count = ((window_size+3) >> 2) + 1;
    while (--count)
    {
        short tmp;
        tmp = (*ptr1 - *ptr2);
        if (tmp > 0)
            amdf += tmp;
        else
            amdf -= tmp;
        ptr1 += 4;
        ptr2 += 4;
    }
    if (amdf < min_amdf)
    {
        best_lag = lag;
        min_amdf = amdf;
    }
}

bs->pitch = lp->olen;
/* Update left buffer */
if (best_lag < (lp->olen >> 1))
{
    /* Add best_lag samples to the length of left pulse*/
    lp->olen += best_lag;
}
else
{
    /* Delete a few samples from the left pulse */
```

```
                lp->olen = best_lag;
            }
            bs->lptr = best_lag;
            weight_inc = 32767/ window_size;
            weight = 32767 - weight_inc;

            ptr1 = rp->bufp;
            ptr2 = lp->bufp + bs->lptr;
            count = window_size+1;
            while (--count)
            {
                *ptr1++ += (((short) (*ptr2++ - *ptr1) * weight) >> 15);
                weight -= weight_inc;
            }
        }
    }
    else
    {
        register short    delta;

        /* Just blend 15 samples */
        ptr2 = lp->bufp + lp->olen - 15;
        ptr1 = rp->bufp;
/*
        for (i = 1; i < 16; i++)
        {
            *ptr1 = *ptr2 + (i * (*ptr1 - *ptr2)) >> 4;
            ptr1++;
            ptr2++;
        }
*/
        delta = *ptr1 - *ptr2;
        *ptr1++ = *ptr2++ + (delta >> 4);

        delta = *ptr1 - *ptr2;
        *ptr1++ = *ptr2++ + ((delta) >> 3);

        delta = *ptr1 - *ptr2;
        *ptr1++ = *ptr2++ + ((3 * delta) >> 4);

        delta = *ptr1 - *ptr2;
        *ptr1++ = *ptr2++ + (delta >> 2);

        delta = *ptr1 - *ptr2;
        *ptr1++ = *ptr2++ + ((5 * delta) >> 4);

        delta = *ptr1 - *ptr2;
        *ptr1++ = *ptr2++ + ((3 * delta) >> 8);

        delta = *ptr1 - *ptr2;
```

```
        *ptr1++ = *ptr2++ + ((7 * delta) >> 4);

        delta = *ptr1 - *ptr2;
        *ptr1++ = *ptr2++ + (delta >> 1);

        delta = *ptr1 - *ptr2;
        *ptr1++ = *ptr2++ + (((delta << 3) + delta) >> 4);

        delta = *ptr1 - *ptr2;
        *ptr1++ = *ptr2++ + ((5 * delta) >> 3);

        delta = *ptr1 - *ptr2;
        *ptr1++ = *ptr2++ + (((delta << 3) + 3 * delta) >> 4);

        delta = *ptr1 - *ptr2;
        *ptr1++ = *ptr2++ + ((3 * delta) >> 2);

        delta = *ptr1 - *ptr2;
        *ptr1++ = *ptr2++ + (((delta << 4) - 3 * delta) >> 4);

        delta = *ptr1 - *ptr2;
        *ptr1++ = *ptr2++ + ((7 * delta) >> 3);

        delta = *ptr1 - *ptr2;
        *ptr1 = *ptr2 + (((delta << 4) - delta) >> 4);


        lp->olen -= 15;
    }
}
```

IV. INTONATION ADJUSTMENT MODULE

```
/* A module for deleting a pitch period */
/*
   Pointer src1 points to Left Pitch period
   Pointer src2 points to Right Pitch period
   Pointer dst points to Resulting Pitch period
   len = length of the pitch periods
*/
skip_pulses(short *src1, short *src2, short *dst, short len)
{
   short i;
   register short    weight, cweight;

   i = len+1;
   weight = cweight = 32767/i;
   while (--i)
   {
       *dst++ = *src1++ + (((short) (*src2++ - *src1) * cweight) >> 15);
       cweight += weight;
   }
}

/* A module for Inserting a pitch period */
/*
   Locn buffer[curbeg] points to Left Pitch period
   Locn buffer[curbeg+curlen] points to Right Pitch period
   Pointer dst points to Resulting Pitch period
   curlen = length of the pitch periods
*/
insert_pulse(short *buffer, short *dst, short curlen,  short curbeg)
{
   short    weight, cweight, count;
   short    *src1, *src2;

   src1 = buffer + curbeg;
   src2 = buffer + curbeg + curlen;
   weight = 32767 / curlen;
   cweight = weight;
   count = curlen + 1;
   while (--count)
   {
       *dst++ = *src1++ = *src2++ + (((short) (*src1 - *src2) * cweight) >>
15);
       cweight += weight;
   }
}

   /* This module is used to change pitch information in the concatenated speech */
```

```
// This routine depends on the desired length (deslen) being at least half
// and no more than twice the actual length (len).

void SnChangePitch(short *buf, short *next, short len, short deslen,short lvoc,short
rvoc,short dosmooth)
{
#pragma unused(rvoc, dosmooth)
    short   delta;
    short   count;
    short   *bptr, *aptr;
    short   weight, weight_inc;
    if (!lvoc || (deslen == len)) return;

    if (deslen > len)
    {
        /* Increase Pitch period */
        delta = deslen - len;
        bptr = buf + len;
        aptr = buf + deslen;
        count = delta + 1;
        while (--count)
            *--aptr = *--bptr;

        count = len - delta + 1;
        weight = weight_inc = 32767 / count;
        while (--count)
        {
            register short tmp2;
            tmp2 = (*--aptr - *--bptr);
            *aptr = *bptr + ((tmp2 * weight) >> 15);
            weight += weight_inc;
        }
        return;
    }
    {
        /* Shorten Pitch Period */
        short wsize;

        delta = len - deslen;
        wsize = 2 * delta;

        if (wsize > deslen)
            wsize = deslen;

        weight_inc = 32767 / (wsize + 1);
        weight = weight_inc;
        aptr = buf + deslen;
        bptr = buf + len - wsize;
        count = wsize - delta + 1;
```

```
            while (--count)
            {
                *bptr++ += (((short) (*aptr++ - *bptr) * weight ) >> 15);
                weight += weight_inc;
            }
            aptr = buf + deslen;
            bptr = next;
            count = delta + 1;
            weight = 32767 - weight;
            while (--count)
            {
                *bptr++ += (((short) (*aptr++ - *bptr) * weight ) >> 15);
                weight -= weight_inc;
            }
        }
    }
```

03.12.99

694 20 547.8 F15004EP/DE

Apple Computer, Inc. 03.12.99/sh/tk

Patentansprüche

1. Vorrichtung zur Verkettung eines ersten digitalen Rahmens von N Proben mit jeweiligen Beträgen, welche eine erste quasiperiodische Wellenform darstellen, und eines zweiten digitalen Rahmens von M Proben mit jeweiligen Beträgen, bzw. Amplituden welche eine zweite quasiperiodische Wellenform darstellen, mit:
   - einem Puffer (15) zum Speichern der Proben des ersten und zweiten digitalen Rahmens;
   - Mitteln, welche mit dem Pufferspeicher gekoppelt sind, zur Bestimmung eines Mischungspunktes für den ersten und den zweiten digitalen Rahmen, ansprechend auf die Beträge der Proben in dem ersten und dem zweiten digitalen Rahmen;
   - Vermischungsmitteln, welche mit dem Pufferspeicher und den Mitteln zur Bestimmung gekoppelt sind, zur Berechnung einer digitalen Sequenz, welche eine Verkettung der ersten und der zweiten quasiperiodischen Wellenform ansprechend auf den ersten Rahmen, den zweiten Rahmen und den Vermischungspunkt darstellt.

2. Vorrichtung nach Anspruch 1, ferner mit:
   - Wandlermitteln, welche mit den Vermischungsmitteln gekoppelt sind, zum Wandeln der digitalen Sequenz in eine analoge verkettete Wellenform.

3. Vorrichtung nach einem der Ansprüche 1 oder 2, bei welcher die Mittel zur Bestimmung aufweisen:

- erste Mittel zur Berechnung eines erweiterten Rahmens ansprechend auf den ersten digitalen Rahmen;
- zweite Mittel zum Auffinden einer Teilmenge des erweiterten Rahmens, welche bezüglich des zweiten digitalen Rahmens relativ gut angepaßt ist, und zur Definierung des Vermischungspunktes als einer Probe in der Teilmenge.

4. Vorrichtung nach Anspruch 3, bei welcher der erweiterte Rahmen eine Verkettung des ersten digitalen Rahmens mit einer Kopie des ersten digitalen Rahmens aufweist.

5. Vorrichtung nach einem der Ansprüche 3 oder 4, bei welcher die Teilmenge des erweiterten Rahmens, welche bezüglich des zweiten digitalen Rahmens relativ gut angepaßt ist, ein Teilmenge mit einer minimalen mittleren bzw. durchschnittlichen Betragsdifferenz über die Proben in der Teilmenge ist, und der Vermischungspunkt eine erste Probe in der Teilmenge ist.

6. Vorrichtung nach einem der vorstehenden Ansprüche, bei welcher die Mittel zur Bestimmung aufweisen:
- erste Mittel zur Berechnung eines erweiterten Rahmens mit einer diskontinuitätsgeglätteten Verkettung des ersten digitalen Rahmens mit einer Kopie des ersten digitalen Rahmens;
- zweite Mittel zum Auffinden einer Teilmenge des erweiterten Rahmens mit einer minimalen durchschnittlichen Betragsdifferenz zwischen den Proben in der Teilmenge und dem zweiten digitalen Rahmen, und zur Definierung eines Vermischungspunktes als einer ersten Probe in der Teilmenge.

7. Vorrichtung nach einem der vorstehenden Ansprüche, bei welcher die Vermischungsmittel aufweisen:

- Mittel zur Zur-Verfügung-Stellung einer ersten Menge von Proben abgeleitet von dem ersten digitalen Rahmen und dem Vermischungspunkt als ein erstes Segment der digitalen Sequenz; und
- Mittel zur Kombination des zweiten digitalen Rahmens mit einer zweiten Menge von Proben, welche von dem ersten digitalen Rahmen und dem Vermischungspunkt abgeleitet sind, unter Betonung der zweiten Menge in einer Startprobe und Betonung des zweiten digitalen Rahmens in einer Endprobe zur Herstellung eines zweiten Segmentes der digitalen Sequenz.

8. Vorrichtung nach Anspruch 6, bei welcher die Vermischungsmittel aufweisen:

- Mittel zur Zur-Verfügung-Stellung einer ersten Menge von Proben, welche von dem ersten digitalen Rahmen und dem Vermischungspunkt abgeleitet sind als ein erstes Segment der digitalen Sequenz; und
- Mittel zur Kombination des zweiten digitalen Rahmens mit der Teilmenge des erweiterten Rahmens, unter Betonung der Teilmenge des erweiterten Rahmens in einer Anfangsprobe und Betonung des zweiten digitalen Rahmens in einer Endprobe zur Herstellung eines zweiten Segmentes der digitalen Sequenz.

9. Vorrichtung nach Anspruch 8, bei welcher der erste und der zweite digitale Rahmen Enden bzw. Anfänge von benachbarten Diphonen bei der Sprache darstellen, und ferner aufweisen:

- Wandlermittel, welche mit den Vermischungsmitteln gekoppelt sind, zum Wandeln der digitalen Sequenz in einen Laut bei der Sprachsynthese.

10. Vorrichtung zur Verkettung eines ersten digitalen Rahmens von N Proben mit jeweiligen Beträgen, welche ein erstes Lautsegment darstellen, und eines zweiten digitalen Rahmens von M Proben mit jeweiligen Beträgen, welche ein zweites Lautsegment darstellen, mit:

- einem Pufferspeicher zum Speichern der Proben des ersten und des zweiten digitalen Rahmens;
- Mitteln, welche mit dem Pufferspeicher gekoppelt sind, zur Bestimmung eines Vermischungspunktes für den ersten und den zweiten digitalen Rahmen ansprechend auf die Beträge der Proben in dem ersten und dem zweiten digitalen Rahmen;
- Vermischungsmitteln, welche mit dem Pufferspeicher und den Mitteln zur Bestimmung gekoppelt sind, zur Berechnung einer digitalen Sequenz, welche eine Verkettung der ersten und der zweiten Lautsegmente ansprechend auf den ersten Rahmen, den zweiten Rahmen und den Vermischungspunkt darstellt; und
- Wandlermitteln, welche mit den Vermischungsmitteln gekoppelt sind, zum Wandeln der digitalen Sequenz in Laute.

11. Vorrichtung nach Anspruch 10, bei welcher die Mittel zur Bestimmung aufweisen:

- erste Mittel zur Berechnung eines erweiterten Rahmens ansprechend auf den ersten digitalen Rahmen;
- zweite Mittel zum Auffinden einer Teilmenge des erweiterten Rahmens, welche bezüglich des zweiten digitalen Rahmens relativ gut angepaßt ist, und zur Definierung

des Vermischungspunktes als einer Probe in der Teilmenge.

12. Vorrichtung nach Anspruch 11, bei welcher der erweiterte Rahmen eine Verkettung des ersten digitalen Rahmens mit einer Kopie des ersten digitalen Rahmens aufweist.

13. Vorrichtung nach einem der Ansprüche 11 oder 12, bei welcher die Teilmenge des erweiterten Rahmens, welche bezüglich des zweiten digitalen Rahmens relativ gut angepaßt ist, eine Teilmenge mit einer minimalen durchschnittlichen Betragsdifferenz über die Proben in der Teilmenge ist, und der Vermischungspunkt eine erste Probe in der Teilmenge ist.

14. Vorrichtung nach einem der Ansprüche 10 bis 13, wobei die Mittel zur Bestimmung aufweisen:

- erste Mittel zur Berechnung eines erweiterten Rahmens mit einer diskontinuitätsgeglätteten Verkettung des ersten digitalen Rahmens mit einer Kopie des ersten digitalen Rahmens;
- zweite Mittel zum Auffinden einer Teilmenge des erweiterten Rahmens mit einer minimalen durchschnittlichen Betragsdifferenz zwischen den Proben in der Teilmenge und dem zweiten digitalen Rahmen und zur Definierung des Vermischungspunktes als einer ersten Probe in der Teilmenge.

15. Vorrichtung nach einem der Ansprüche 10 bis 14, wobei die Vermischungsmittel aufweisen:

- Mittel zur Zur-Verfügung-Stellung einer ersten Menge von Proben, welche von dem ersten digitalen Rahmen und dem

Vermischungspunkt abgeleitet sind, als ein erstes Segment der digitalen Sequenz; und

- Mittel zur Kombination des zweiten digitalen Rahmens mit einer zweiten Menge von Proben, die von dem ersten digitalen Rahmen und dem Vermischungspunkt abgeleitet sind, mit Betonung der zweiten Menge in einer Startprobe und Betonung des zweiten digitalen Rahmens in einer Endprobe zur Herstellung eines zweiten Segments der digitalen Sequenz.

16. Vorrichtung nach Anspruch 14, bei welcher die Vermischungsmittel aufweisen:

- Mittel zur Zur-Verfügung-Stellung einer ersten Menge von Proben, die von dem ersten digitalen Rahmen und dem Vermischungspunkt abgeleitet sind, als ein erstes Segment der digitalen Sequenz; und
- Mittel zur Kombination des zweiten digitalen Rahmens mit der Teilmenge des erweiterten Rahmens, mit Betonung der Teilmenge des erweiterten Rahmens in einer Startprobe und Betonung des zweiten digitalen Rahmens in einer Endprobe zur Herstellung eines zweiten Segmentes der digitalen Sequenz.

17. Vorrichtung nach Anspruch 16, bei welcher der erste und der zweite digitale Rahmen Enden bzw. Anfänge benachbarter Diphone in der Sprache darstellen, und die Wandlermittel synthetisierte Sprache erzeugen.

18. Vorrichtung zur Synthetisierung von Sprache ansprechend auf einen Text, mit

- Mitteln (21) zur Übersetzung von Text in eine Sequenz von Lautsegmentcodierungen;
- Mitteln (23), welche ansprechend sind auf die Lautsegmentcodierungen in der Sequenz, zur Decodierung der Se-

quenz der Lautsegmentcodierungen zur Herstellung von Strings von digitalen Rahmen einer Anzahl von Proben, welche Laute für jeweilige Lautsegmentcodierungen in der Sequenz darstellen, wobei die identifizierten Strings der digitalen Rahmen Anfänge und Endungen bwz. Enden aufweisen;

- Mitteln (24) zur Verkettung eines ersten digitalen Rahmens an der Endung eines identifizierten Strings von digitalen Rahmen einer bestimmten Lautsegmentcodierung in den Sequenzen mit einem zweiten digitalen Rahmen am Anfang eines identifizierten Strings von digitalen Rahmen einer benachbarten Lautsequenzcodierung in der Sequenz zur Erzeugung einer Sprachdatensequenz, mit

  einem Pufferspeicher zum Speichern der Proben von ersten und zweiten digitalen Rahmen;

  Mitteln, welche mit dem Pufferspeicher gekoppelt sind, zur Bestimmung eines Vermischungspunktes für den ersten und den zweiten digitalen Rahmen, ansprechend auf die Beträge der Proben in dem ersten und dem zweiten digitalen Rahmen;

  Vermischungsmitteln, welche mit dem Pufferspeicher und den Mitteln zur Bestimmung gekoppelt sind, zur Berechnung einer digitalen Sequenz, welche eine Verkettung der ersten und zweiten Lautsegmente ansprechend auf den ersten Rahmen, den zweiten Rahmen und den Vermischungspunkt darstellt; und

  einem Audiowandler (27), der mit den Mitteln zur Verkettung gekoppelt ist, zur Generierung synthetisierter Sprache ansprechend auf die Sprachdatensequenz.

19. Vorrichtung nach Anspruch 18, ferner mit:

- Mitteln, welche ansprechend auf die Lautsegmentcodierungen sind, zur Einstellung der Tonhöhe und der Dauer der

identifizierten Strings der digitalen Rahmen in der Sprachdatensequenz.

20. Vorrichtung nach einem der Ansprüche 18 oder 19, bei welcher die Mittel zur Bestimmung aufweisen:

- erste Mittel zur Berechnung eines erweiterten Rahmens ansprechend auf den ersten digitalen Rahmen;
- zweite Mittel zum Auffinden einer Teilmenge des erweiterten Rahmens, welcher bezüglich des zweiten digitalen Rahmens relativ gut angepaßt ist und zur Definierung des Vermischungspunktes als einer Probe in der Teilmenge.

21. Vorrichtung nach Anspruch 20, bei welcher der erweiterte Rahmen eine Verkettung des ersten Rahmens mit einer Kopie des ersten digitalen Rahmens aufweist.

22. Vorrichtung nach einem der Ansprüche 20 oder 21, bei welcher die Teilmenge des erweiterten Rahmens, welche bezüglich des ersten digitalen Rahmens relativ gut angepaßt ist, eine Teilmenge mit einer minimalen durchschnittlichen Betragsdifferenz über die Proben in der Teilmenge aufweist, und wobei der Vermischungspunkt eine erste Probe in der Teilmenge aufweist.

23. Vorrichtung nach einem der Ansprüche 18 bis 22, bei welcher die Mittel zur Bestimmung aufweisen:

- erste Mittel zur Berechnung eines erweiterten Rahmens mit einer diskontinuitätsgeglätteten Verkettung des ersten digitalen Rahmens mit einer Kopie des ersten digitalen Rahmens;
- zweite Mittel zum Auffinden einer Teilmenge des erweiterten Rahmens mit einer minimalen durchschnittlichen Betragsdifferenz zwischen den Proben in der Teilmenge

und dem zweiten digitalen Rahmen, und zur Definierung des Vermischungspunktes als einer ersten Probe in der Teilmenge.

24. Vorrichtung nach einem der Ansprüche 18 bis 23, bei welcher die Vermischungsmittel aufweisen:

- Mittel zur Zur-Verfügung-Stellung einer ersten Menge von Proben, welche von dem ersten digitalen Rahmen und dem Vermischungspunkt abgeleitet sind, als ein erstes Segment der digitalen Sequenz; und
- Mittel zur Kombination des zweiten digitalen Rahmens mit einem zweiten Satz von Proben, welche von dem ersten digitalen Rahmen und dem Vermischungspunkt abgeleitet sind, mit Betonung der zweiten Menge in einer Anfangsprobe und Betonung des zweiten digitalen Rahmens in einer Endprobe zur Herstellung eines zweiten Segmentes der digitalen Sequenz.

25. Vorrichtung nach Anspruch 23, bei welcher die Vermischungsmittel aufweisen:

- Mittel zur Zur-Verfügung-Stellung eines ersten Satzes von Proben, welche abgeleitet sind von dem ersten digitalen Rahmen und dem Vermischungspunkt, als ein erstes Segment der digitalen Sequenz; und
- Mittel zur Kombination des zweiten digitalen Rahmens mit der Teilmenge des erweiterten Rahmens, mit Betonung der Teilmenge des erweiterten Rahmens in einer Anfangsprobe und Betonung des zweiten digitalen Rahmens in einer Endprobe zur Herstellung eines zweiten Segmentes der digitalen Sequenz.

26. Vorrichtung nach einem der Ansprüche 18 bis 25, bei welcher die Lautsegmentcodierungen Sprach-Diphone darstel-

len, und der erste und der zweite digitale Rahmen Endungen bzw. Anfänge benachbarter Diphone in der Sprache darstellen.

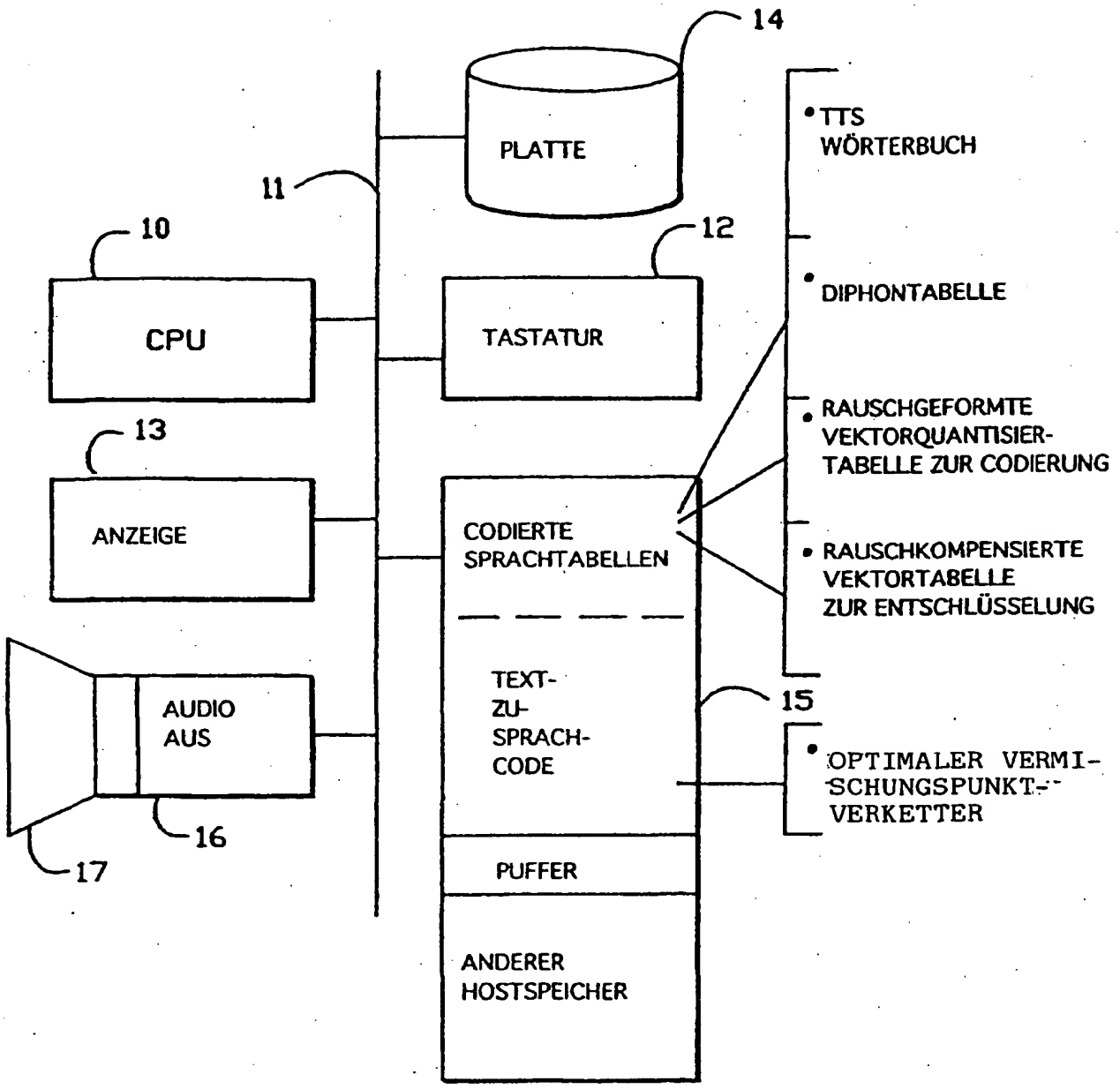

14
PLATTE
11
10
CPU
12
TASTATUR
13
ANZEIGE
AUDIO AUS
16
17
CODIERTE SPRACHTABELLEN
TEXT-ZU-SPRACH-CODE
15
PUFFER
ANDERER HOSTSPEICHER
• TTS WÖRTERBUCH
• DIPHONTABELLE
• RAUSCHGEFORMTE VEKTORQUANTISIER-TABELLE ZUR CODIERUNG
• RAUSCHKOMPENSIERTE VEKTORTABELLE ZUR ENTSCHLÜSSELUNG
• OPTIMALER VERMI-SCHUNGSPUNKT-VERKETTER

FIG.—1

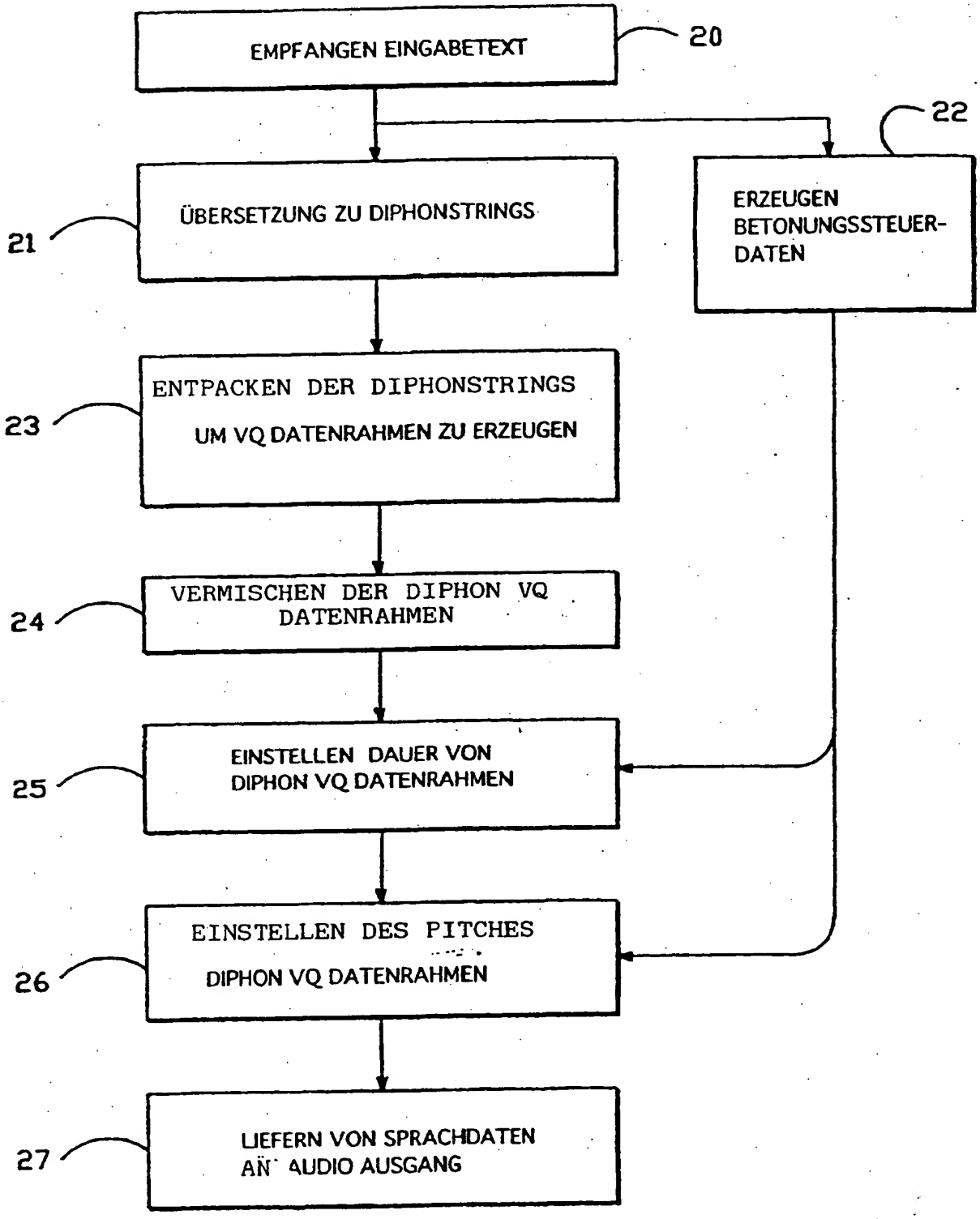

EMPFANGEN EINGABETEXT
20
22
ÜBERSETZUNG ZU DIPHONSTRINGS
21
ERZEUGEN BETONUNGSSTEUER-DATEN
ENTPACKEN DER DIPHONSTRINGS
UM VQ DATENRAHMEN ZU ERZEUGEN
23
VERMISCHEN DER DIPHON VQ DATENRAHMEN
24
EINSTELLEN DAUER VON DIPHON VQ DATENRAHMEN
25
EINSTELLEN DES PITCHES
DIPHON VQ DATENRAHMEN
26
LIEFERN VON SPRACHDATEN AN AUDIO AUSGANG
27
TEXT-ZU-SPRACH-CODE

FIG.—2

DIPHON AUFZEICHNUNG

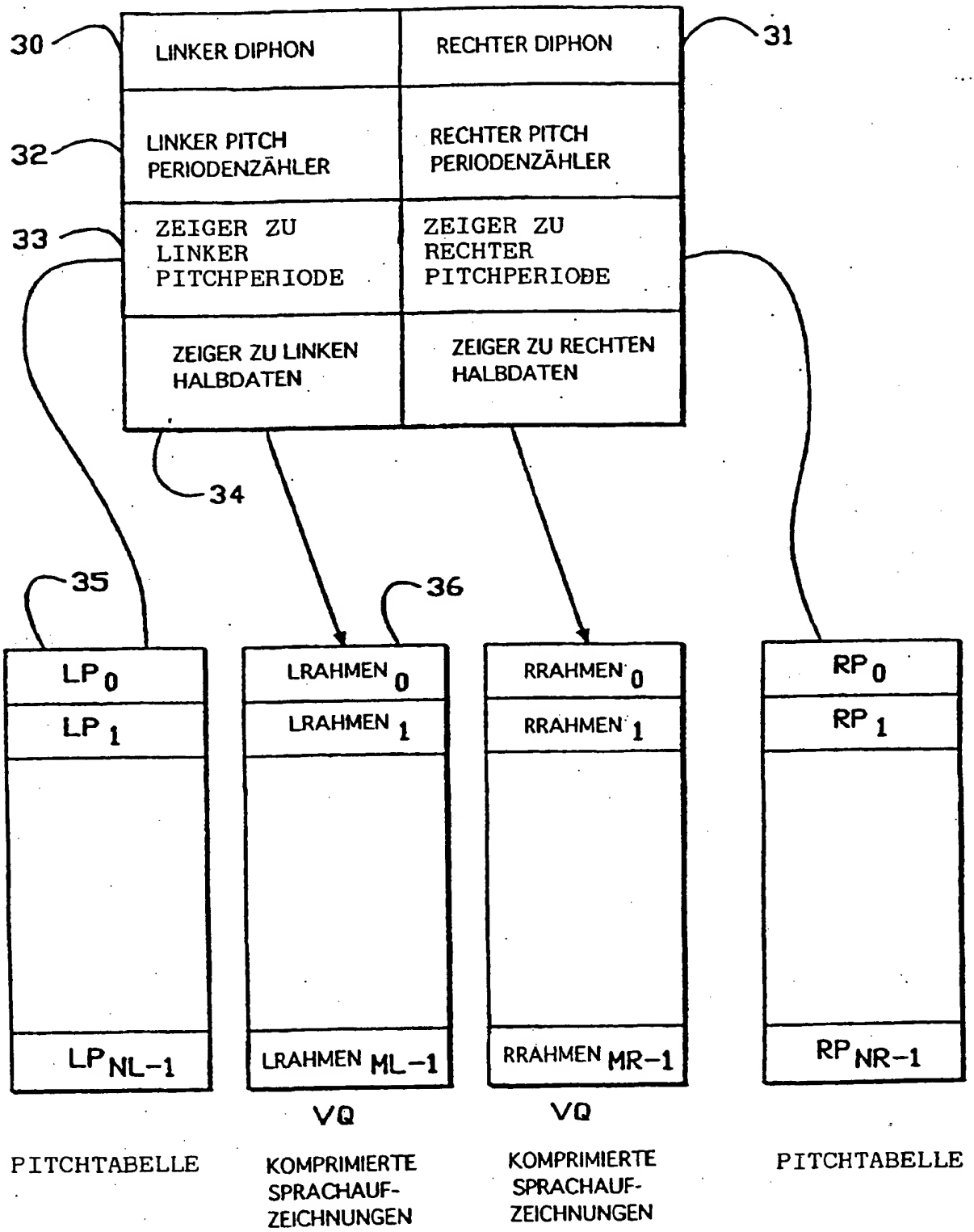


FIG.—3

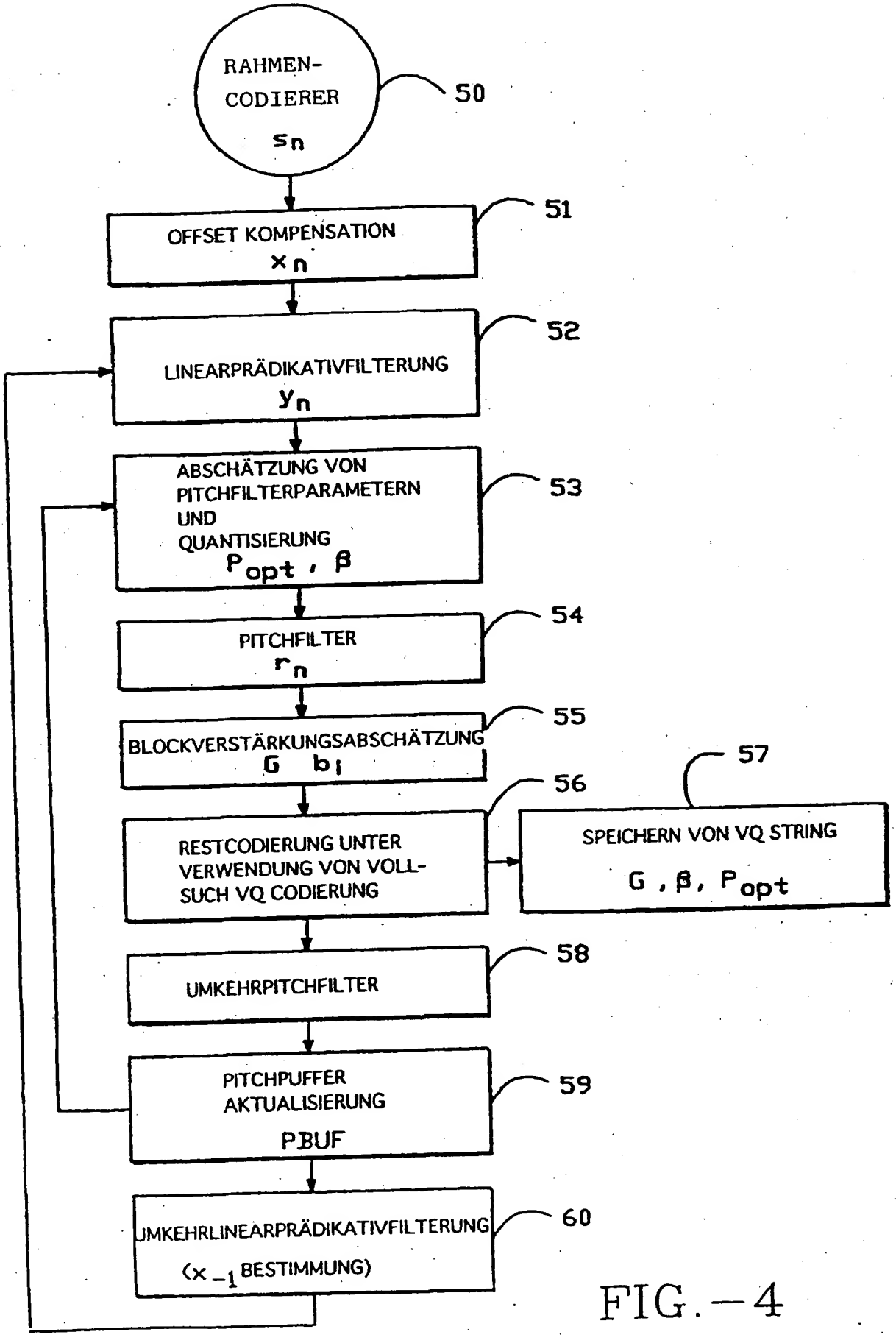

RAHMEN-
CODIERER
s_n
50
OFFSET KOMPENSATION
x_n
51
LINEARPRÄDIKATIVFILTERUNG
y_n
52
ABSCHÄTZUNG VON
PITCHFILTERPARAMETERN
UND
QUANTISIERUNG
P_{opt} , β
53
PITCHFILTER
r_n
54
BLOCKVERSTÄRKUNGSABSCHÄTZUNG
G b_l
55
RESTCODIERUNG UNTER
VERWENDUNG VON VOLL-
SUCH VQ CODIERUNG
56
SPEICHERN VON VQ STRING
G , β, P_{opt}
57
UMKEHRPITCHFILTER
58
PITCHPUFFER
AKTUALISIERUNG
PBUF
59
UMKEHRLINEARPRÄDIKATIVFILTERUNG
(x_{-1} BESTIMMUNG)
60

FIG.—4

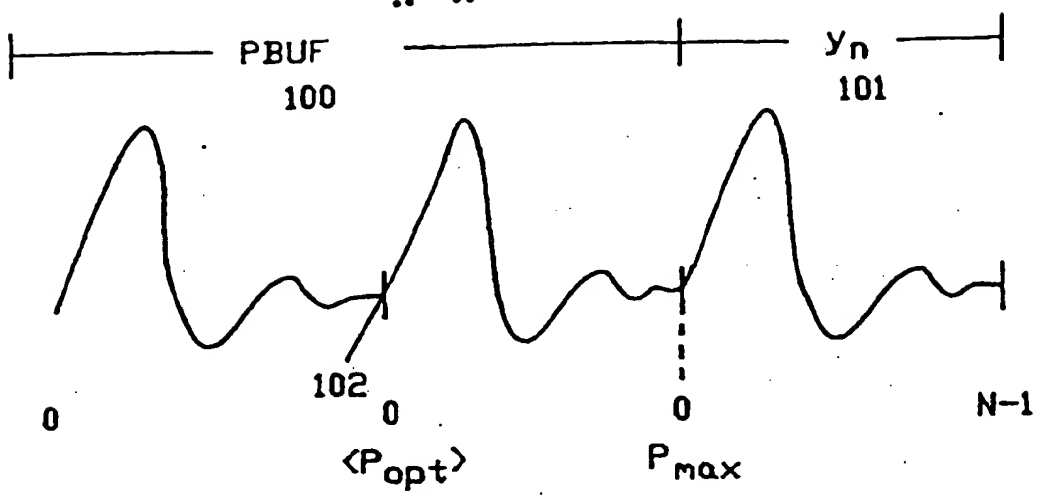


FIG.–5

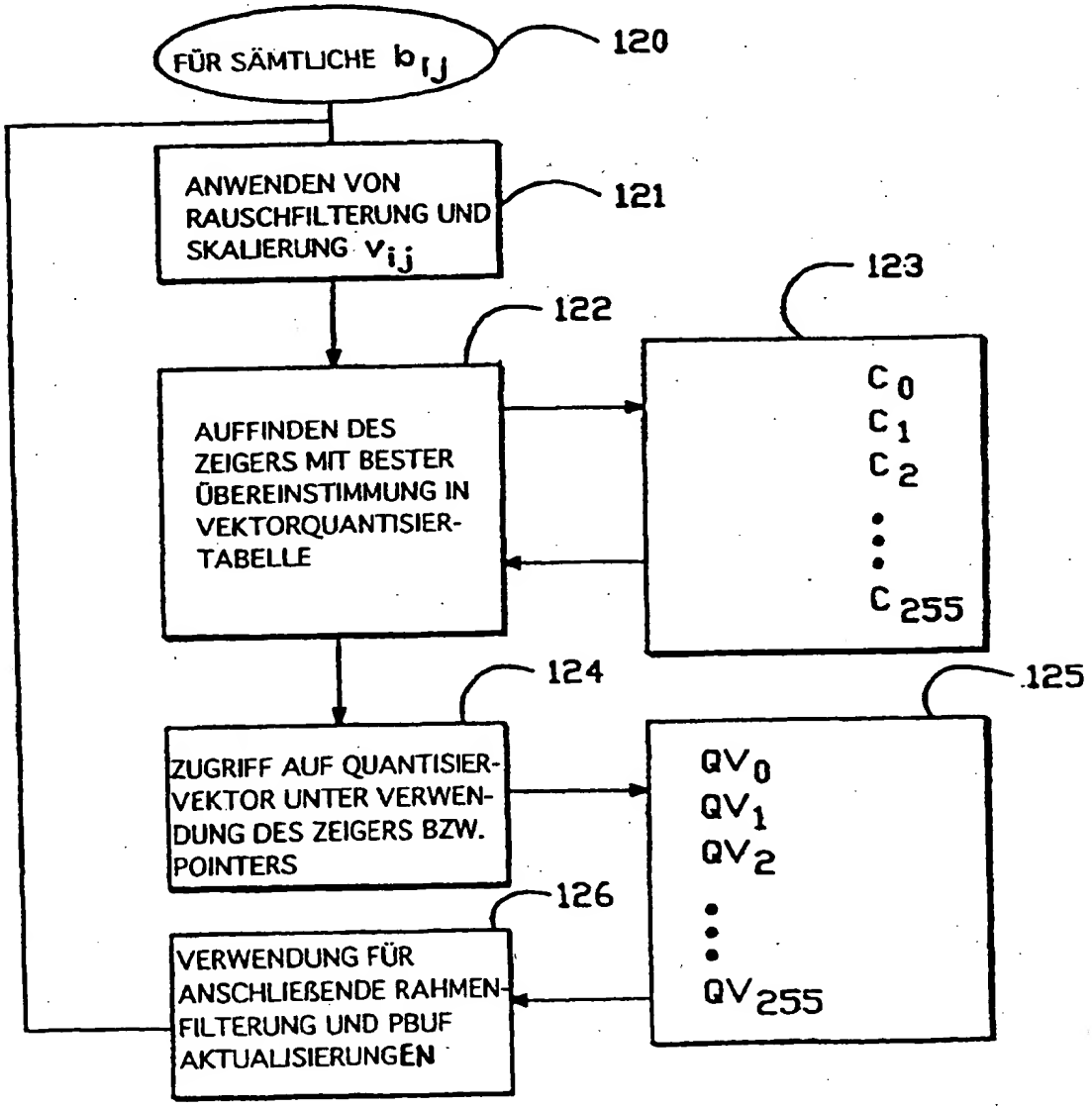


FIG.–6

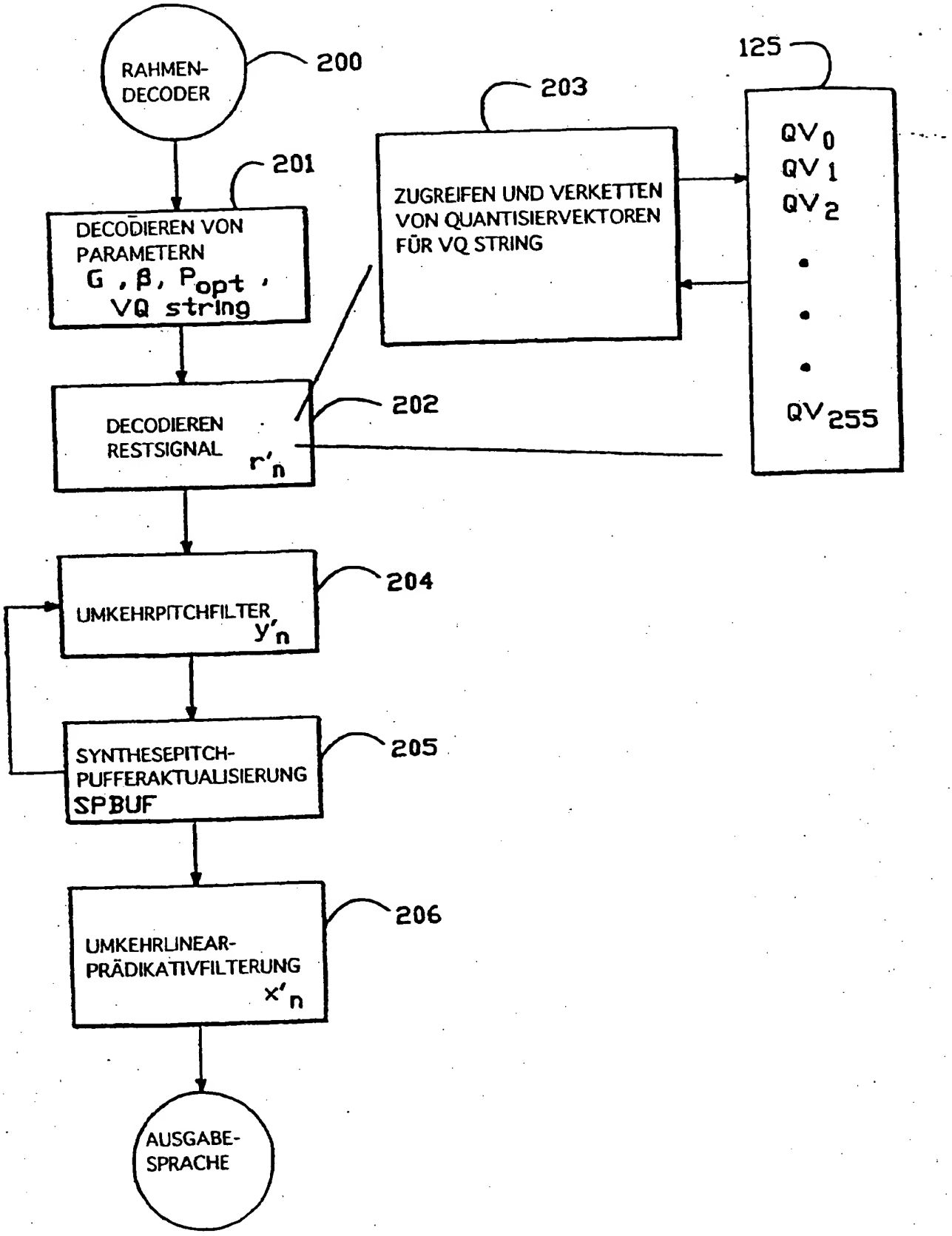

RAHMEN-
DECODER
200
201
DECODIEREN VON
PARAMETERN
G , β, Popt ,
VQ string
203
ZUGREIFEN UND VERKETTEN
VON QUANTISIERVEKTOREN
FÜR VQ STRING
125
QV0
QV1
QV2
QV255
202
DECODIEREN
RESTSIGNAL
r'n
204
UMKEHRPITCHFILTER
y'n
205
SYNTHESEPITCH-
PUFFERAKTUALISIERUNG
SPBUF
206
UMKEHRLINEAR-
PRÄDIKATIVFILTERUNG
x'n
AUSGABE-
SPRACHE

FIG.—7

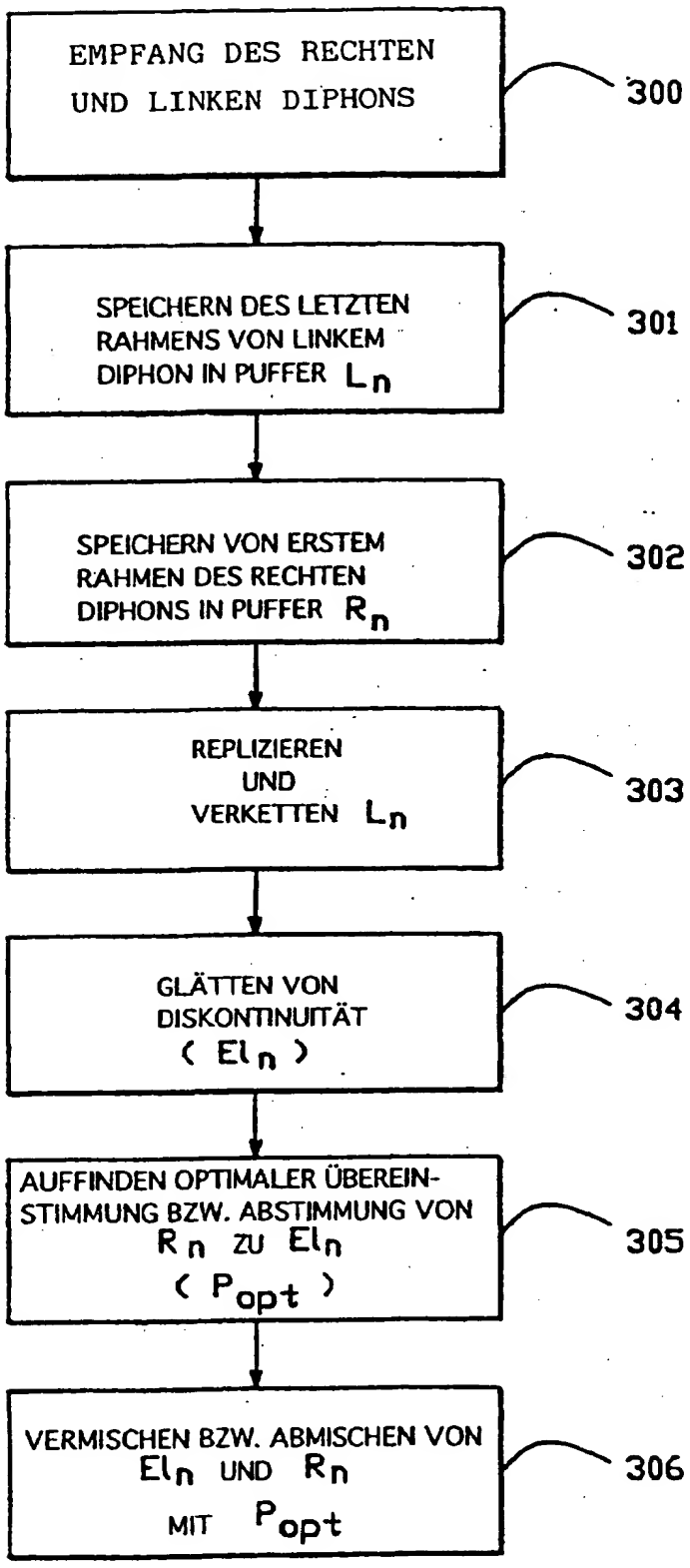

EMPFANG DES RECHTEN UND LINKEN DIPHONS
300
SPEICHERN DES LETZTEN RAHMENS VON LINKEM DIPHON IN PUFFER L_n
301
SPEICHERN VON ERSTEM RAHMEN DES RECHTEN DIPHONS IN PUFFER R_n
302
REPLIZIEREN UND VERKETTEN L_n
303
GLÄTTEN VON DISKONTINUITÄT (El_n)
304
AUFFINDEN OPTIMALER ÜBEREIN-STIMMUNG BZW. ABSTIMMUNG VON R_n ZU El_n (P_{opt})
305
VERMISCHEN BZW. ABMISCHEN VON El_n UND R_n MIT P_{opt}
306

FIG.—8

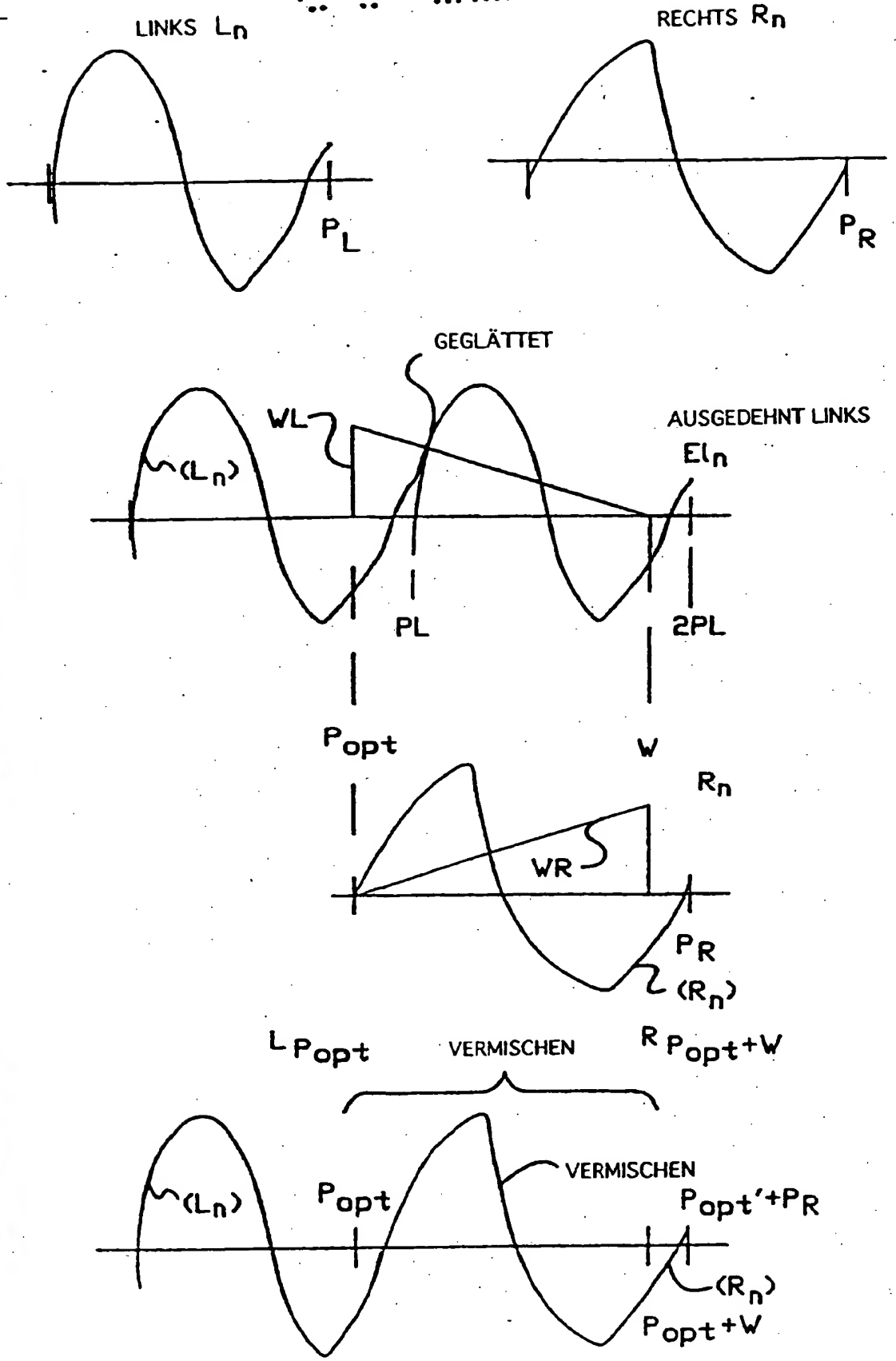
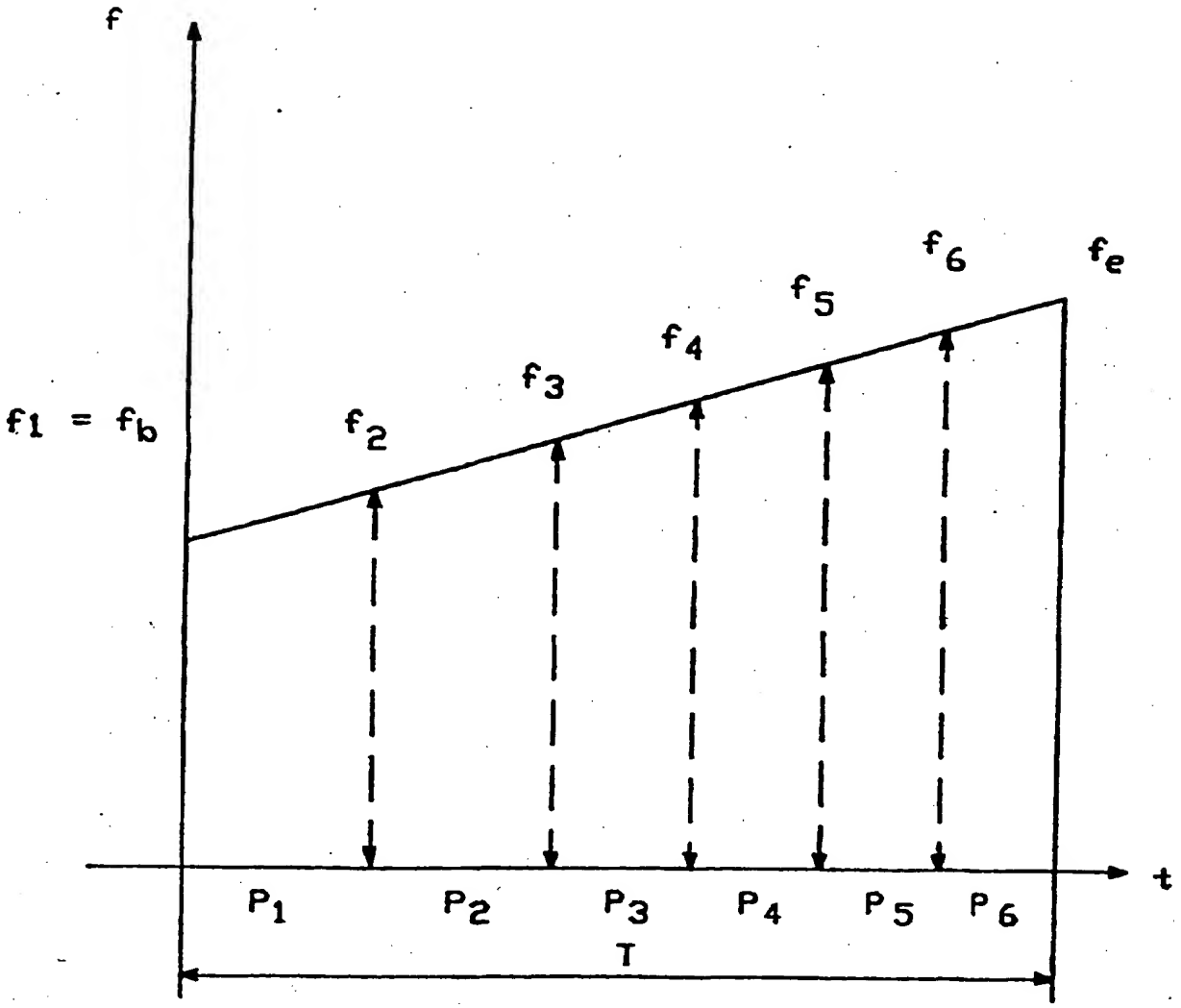

LINKS L_n
P_L
RECHTS R_n
P_R
GEGLÄTTET
WL
AUSGEDEHNT LINKS
El_n
(L_n)
PL
2PL
P_{opt}
W
R_n
WR
P_R
(R_n)
$L_{P_{opt}}$
VERMISCHEN
$R_{P_{opt}+W}$
VERMISCHEN
(L_n)
P_{opt}
$P_{opt'}+P_R$
(R_n)
$P_{opt}+W$

FIG.—9



NOTIZEN:

T = Gewünschte Dauer eines Phonems

f_b = GEWÜNSCHTER Anfangspitch in Hz

f_e = Gewünschter Endungspitch in Hz

P1, P2, ...P6 sind die gewünschten Pitchperioden in der Anzahl von Proben die den Frequenzen f1, f2, ...f6 entspricht.

Beziehung zwischen Pl und fi:

Pi = Fs/fi, wobei Fs die Probennahme bzw. Abtastfrequenz ist.

FIG.—10

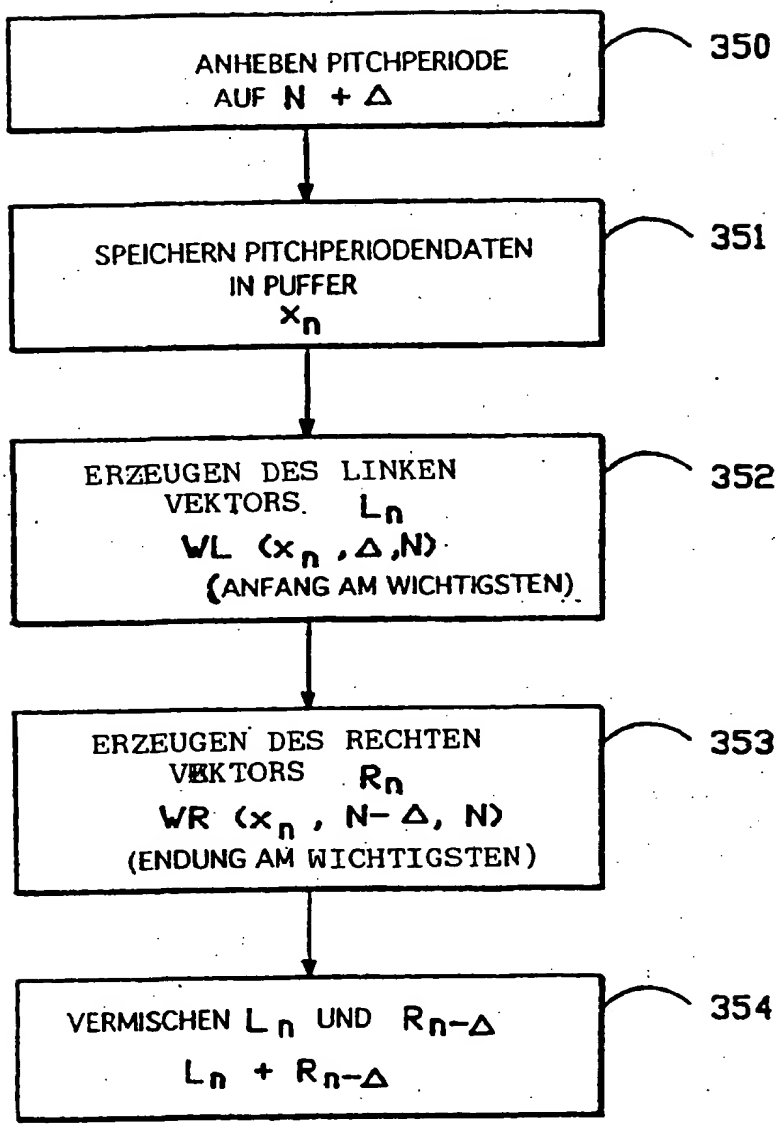


FIG.−11

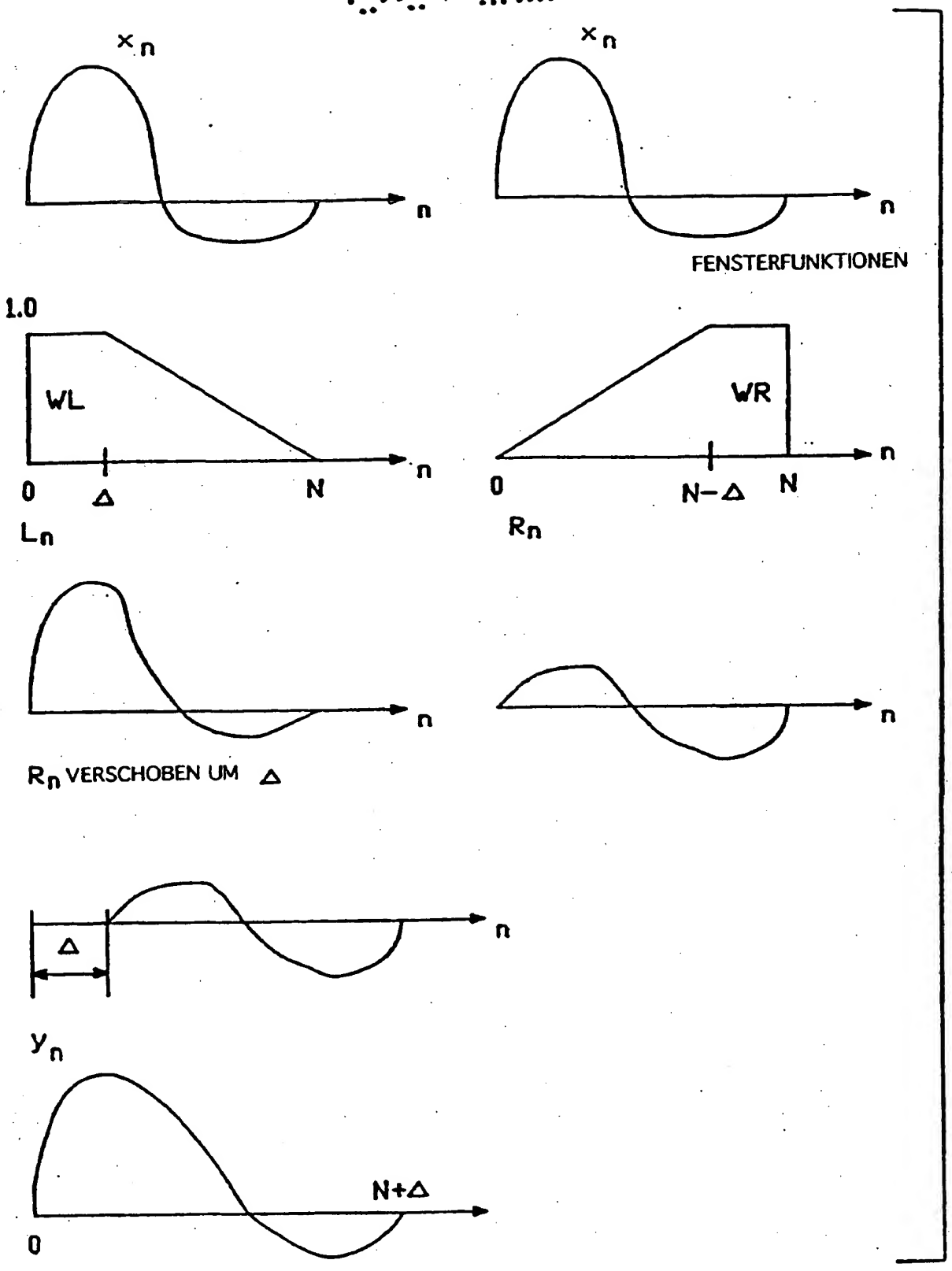


FIG.−12

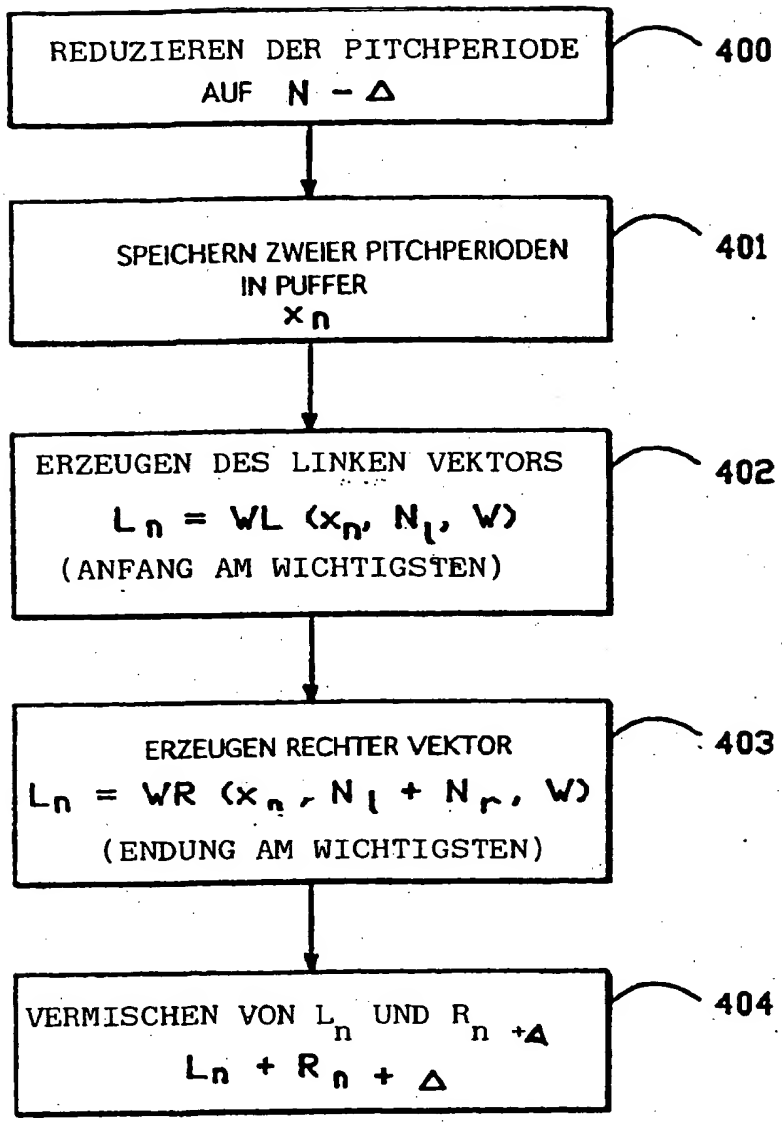


FIG.−13

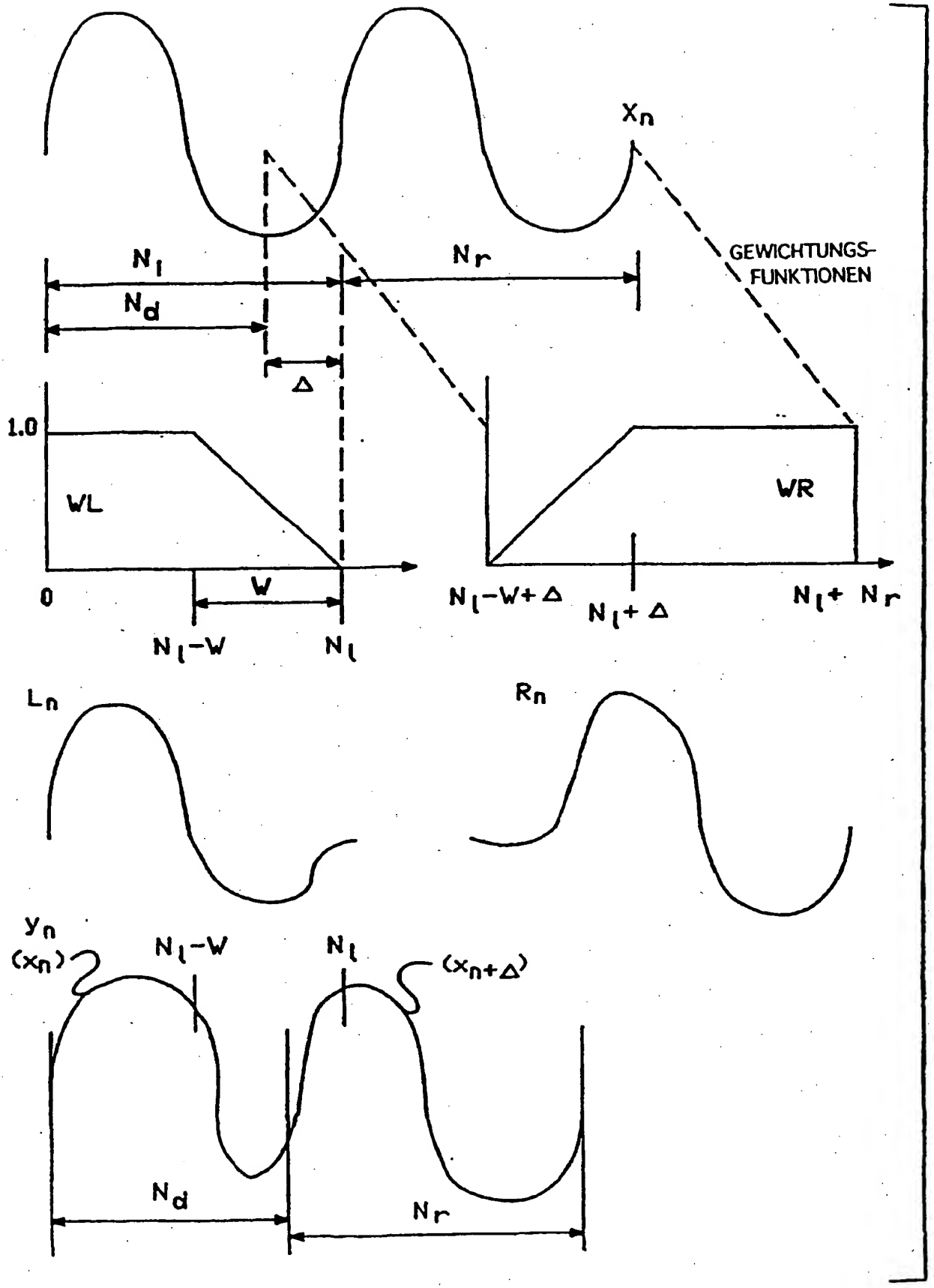


FIG.—14

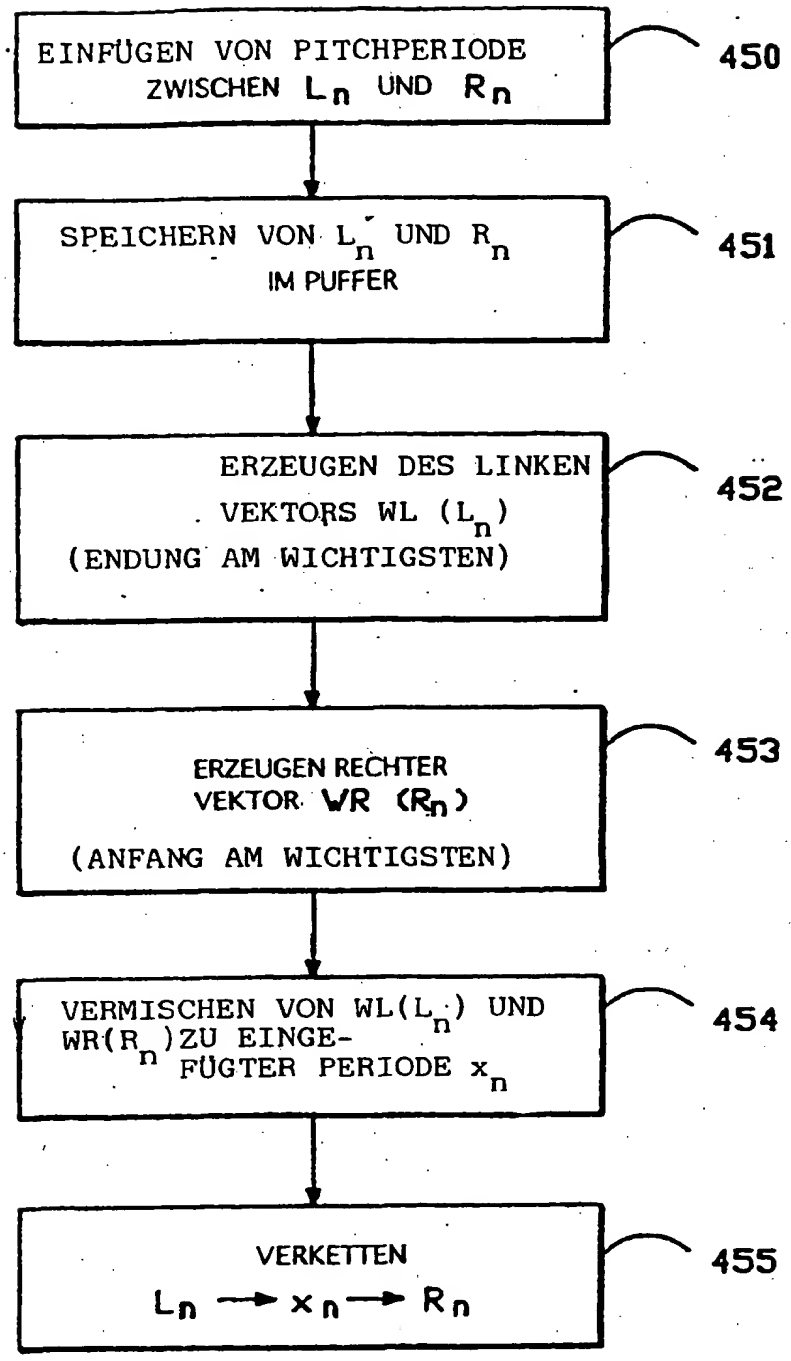


FIG.—15

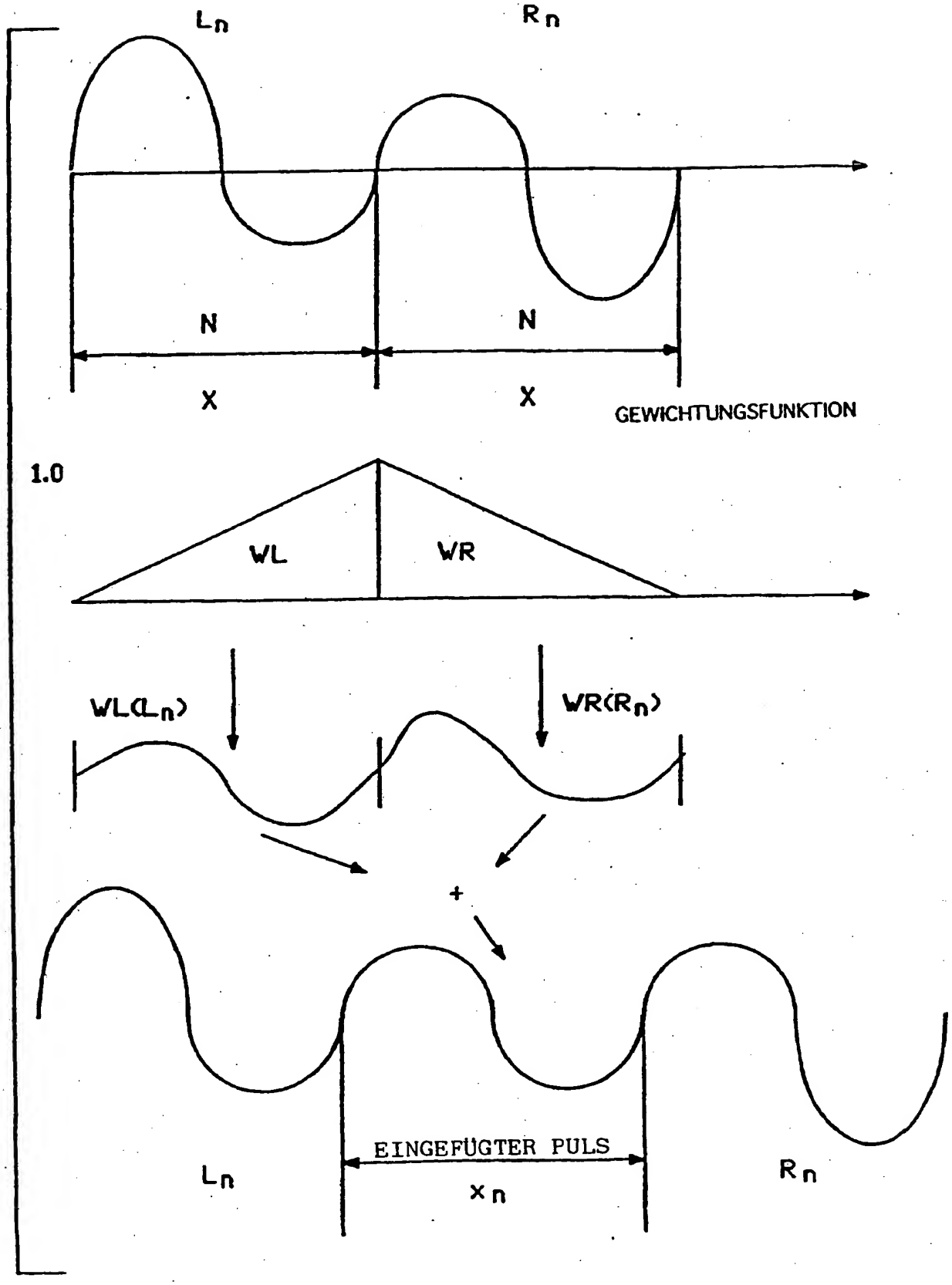


FIG.—16

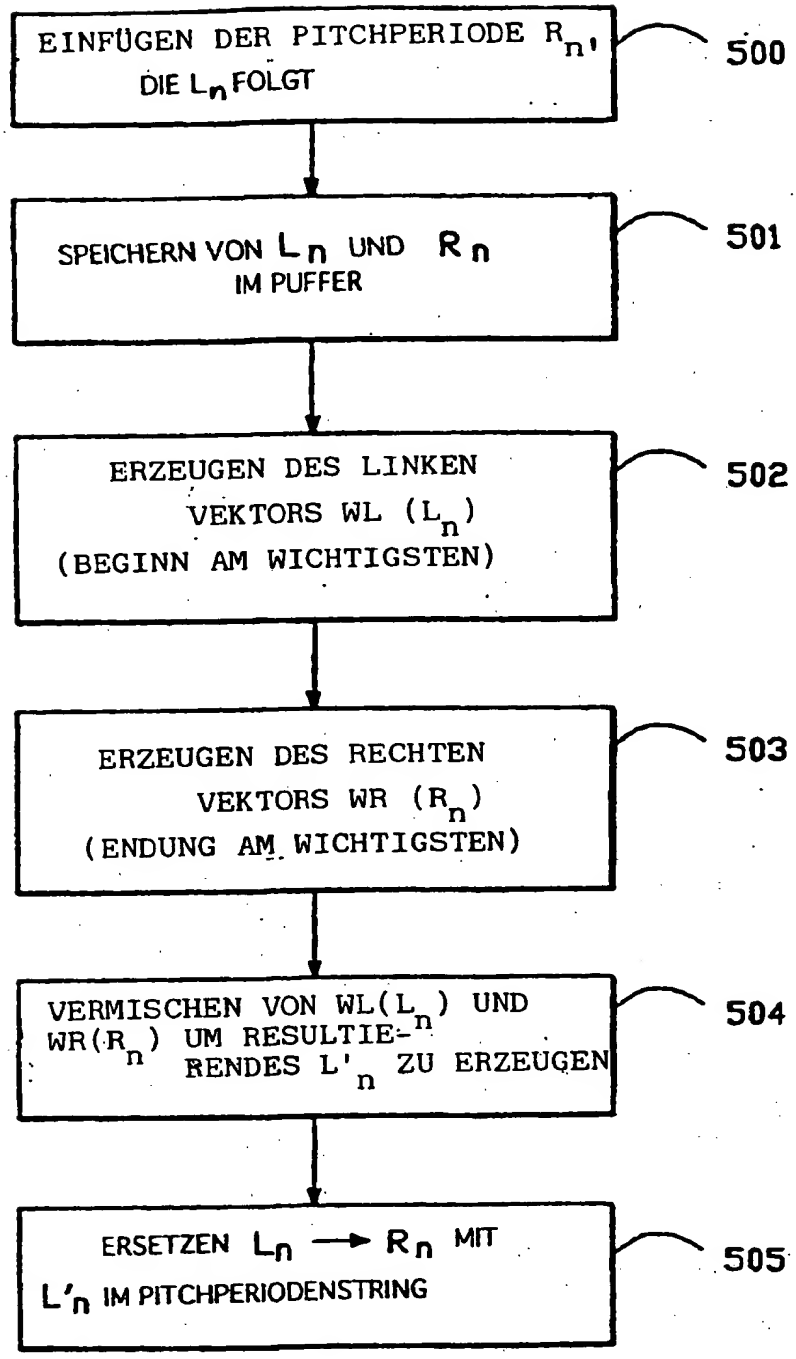


FIG.—17

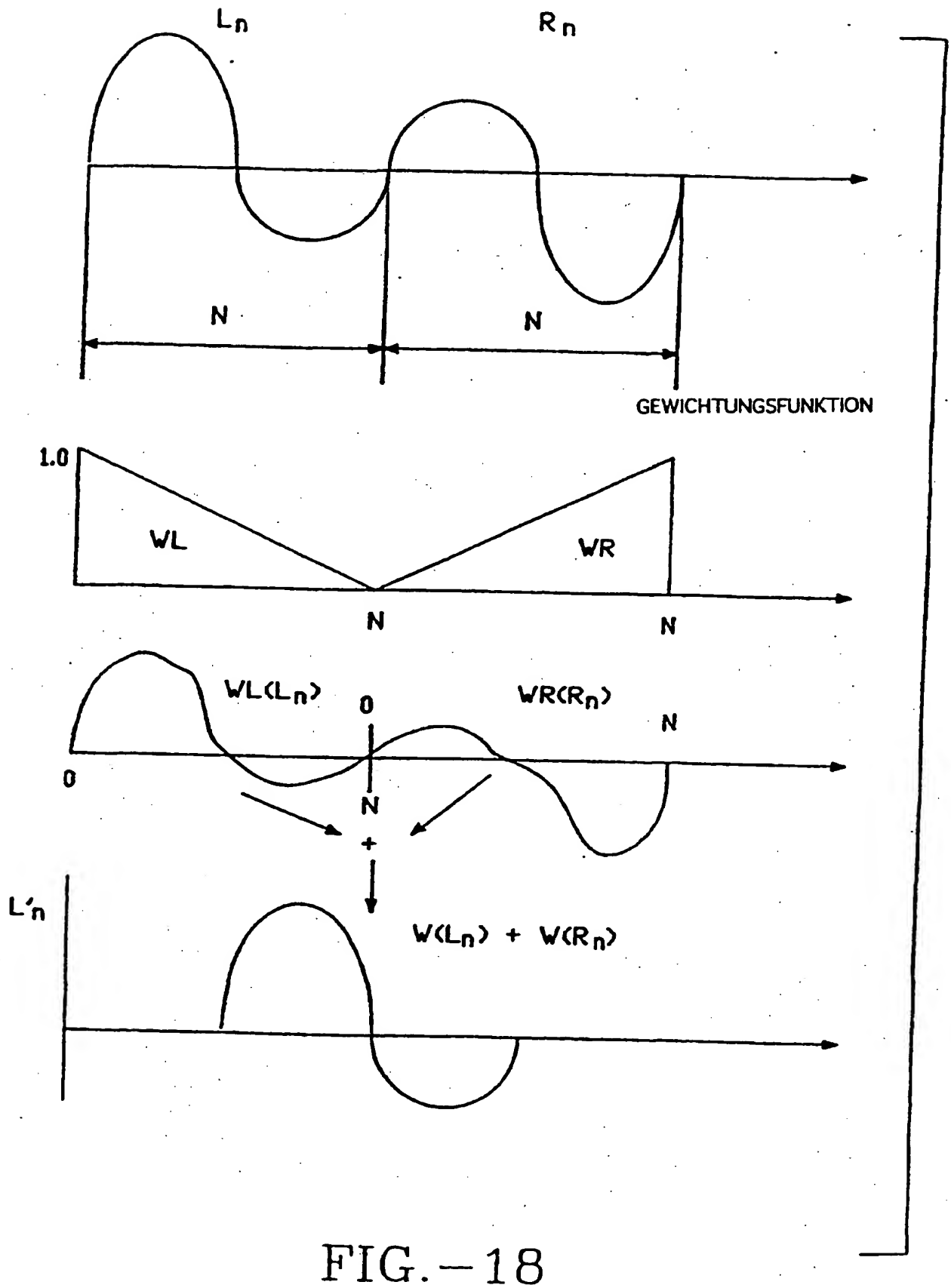


FIG.—18